Wilfrid Laurier University

Scholars Commons @ Laurier

Theses and Dissertations (Comprehensive)

2025

Reinforcement Learning for Scheduling and Routing: Electric Vehicles Charging and Patrol Scheduling

MAJID GHASEMI
Wilfrid Laurier University, ghas1560@mylaurier.ca

Follow this and additional works at: https://scholars.wlu.ca/etd

Part of the Artificial Intelligence and Robotics Commons, and the Theory and Algorithms Commons

Recommended Citation

GHASEMI, MAJID, "Reinforcement Learning for Scheduling and Routing: Electric Vehicles Charging and Patrol Scheduling" (2025). *Theses and Dissertations (Comprehensive)*. 2764.
https://scholars.wlu.ca/etd/2764

This Thesis is brought to you for free and open access by Scholars Commons @ Laurier. It has been accepted for inclusion in Theses and Dissertations (Comprehensive) by an authorized administrator of Scholars Commons @ Laurier. For more information, please contact scholarscommons@wlu.ca.

Reinforcement Learning for Scheduling and Routing: Electric Vehicles Charging and Patrol Scheduling

by

Majid Ghasemi

THESIS

Submitted to the Department of Physics and Computer Science
Faculty of Science
in fulfillment of the requirements for
Master of Applied Computing
Wilfrid Laurier University

Waterloo, Ontario, Canada, 2025

© Majid Ghasemi 2025

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

Supervisor:	Dariush Ebrahimi Asst. Professor, Dept. of Computer Science, Wilfrid Laurier University
Internal Member:	Ilias S. Kotsireas Professor, Dept. of Computer Science, Wilfrid Laurier University
External Examiner:	Thiago E. Alves de Oliveira Asst. Professor, Dept. of Computer Science, Lakehead University

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis offers a comprehensive exploration of Reinforcement Learning (RL), beginning with fundamental theoretical constructs, Markov Decision Processes, Dynamic Programming, Monte Carlo, and Temporal Difference methods, and extending into state-of-the-art deep RL approaches such as Deep Q-Networks (DQN) and policy-gradient algorithms. Through analytical experiments in controlled environments, the thesis demonstrates how distinct algorithmic choices (e.g., exploration techniques, eligibility traces, or network architectures) influence convergence and stability. These foundational insights pave the way for two in-depth case studies, which apply RL techniques to critical, real-world scheduling and routing challenges.

The first case study tackles the Electric Vehicle (EV) routing and charging problem, in which vehicles must navigate a road network populated with limited charging stations under battery and time constraints. To address this, first, a heuristic-based Q-learning (TQL) strategy that extends classical tabular Q-learning with temporal features of traffic and charging demands is proposed. A variant of deep Q-learning (DQL) is then introduced to accommodate larger-scale networks and continuous state parameters. Extensive simulations reveal that RL-based methods, particularly TQL, outperform baseline heuristics by reducing total travel distance and charging overhead. Notably, DQL demonstrates scalability, effectively handling more nodes and dynamic charging station placements, an essential factor for complex, real-time EV routing scenarios.

The second case study dives deeper into fleet management through a vehicle patrol scheduling problem where multiple patrol vehicles must repeatedly traverse a set of critical locations subject to diverse constraints, including patrol frequency, travel times, and emergency calls. This study thoroughly compares classical optimization (e.g., mixed integer linear programming), heuristic (e.g., genetic algorithms, adaptive hill-climbing), and RL-based approaches. Of particular interest is the proposed $\mathcal{P}^4\mathcal{O}$ RL framework, which integrates a policy-based architecture with real-time decision-making to handle unpredictable emergency events and evolving traffic conditions. By combining RL and heuristic strategies, the system maintains efficient patrol coverage while dynamically reallocating vehicles to emergent hotspots. Simulation-based evaluations demonstrate significant improvements in response times, minimized travel distance, and balanced distribution of vehicle workloads, even under high uncertainty. In addition, sensitivity analyses examine how variations in patrol duration, rest times, and emergency rates impact the performance of each algorithm, providing actionable guidelines for deploying RL-driven patrol strategies in real-world settings.

By systematically integrating classical RL algorithms with modern deep learning architectures, this thesis reveals how an agent's learning process can be significantly improved

when critical constraints, such as energy or resource limitations, timing requirements, or rapidly changing system states, are carefully embedded into RL reward structures, state representations, and update mechanisms. Extensive experiments demonstrate that tabular approaches excel in managing smaller, more structured problems, whereas deep methods scale more effectively in complex domains with high-dimensional or continuous state spaces. Crucially, hybrid RL solutions bridge these strengths by incorporating domain-oriented heuristics, adaptive exploration, and targeted function approximation, resulting in faster convergence and more robust behavior in the presence of uncertainty. Collectively, these findings show that thoughtfully designed RL systems, whether tabular, deep, or hybrid, can quickly adapt to new conditions, consistently surpassing conventional heuristics or purely optimization-based methods while maintaining computational feasibility. Ultimately, this work underscores the transformative potential of RL for dynamic scheduling, routing, and resource management, illustrating a practical pathway to achieve higher-quality decision-making in real-world deployments.

Acknowledgements

I wish to express my heartfelt gratitude to those who have supported me throughout this journey.

To myself, I honor the persistence that carried me through the challenges, reminding me of my inner strength and determination.

To my mom and dad, your unconditional support has been my anchor, and your faith in me has been a constant source of motivation. I owe so much of this journey to your love and encouragement.

To my advisor, Dr. Dariush Ebrahimi, I extend my deepest thanks for your exceptional mentorship and unwavering guidance. This thesis would not have been possible without your support. Your expertise and dedication have played a pivotal role in shaping my research and fostering my growth as a scholar. I am profoundly grateful for your invaluable insights and for continuously inspiring me to pursue excellence. You have been the best advisor, exceeding all expectations.

Publications

This thesis comprises a collection of research papers that have either been published in or submitted to a variety of scholarly venues. The following list enumerates these works in detail.

Journal Articles

1. **Title:** "Dynamic and Scalable Vehicle Patrol Scheduling for Urban Safety: Optimization and Heuristic Approaches"
   Authors: Majid Ghasemi, Ibrahim Sorkhoh, Fadi Alzhouri, and Dariush Ebrahimi
   Status: Submitted to *IEEE Transactions on Intelligent Transportation Systems*

2. **Title:** "$\mathcal{P}^4\mathcal{O}$: Real-Time Vehicle Patrol Planning with Proximal Policy Optimization"
   Authors: Majid Ghasemi and Dariush Ebrahimi
   Status: Submitted to *IEEE Transactions on Intelligent Vehicles*

3. **Title:** "A Survey on Deep Reinforcement Learning and Model-based Planning"
   Authors: Majid Ghasemi, Amir Hossein Moosavi, and Dariush Ebrahimi
   Status: Submitted to *IEEE Transactions on Neural Networks and Learning Systems*

4. **Title:** "Foundations of Reinforcement Learning: A Survey of Core Concepts"
   Authors: Majid Ghasemi, Amir Hossein Moosavi, and Dariush Ebrahimi
   Status: Submitted to *IEEE Transactions on Neural Networks and Learning Systems*

Conference Papers

1. **Title:** "Adaptive and AI-Driven Vehicle Patrol Scheduling with Integrated Emergency Response"
   Authors: Majid Ghasemi and Dariush Ebrahimi
   Status: Submitted to *IEEE Smart World Congress 2025*

2. **Title:** "Optimal and Heuristic Solutions for Efficient Vehicle Patrol Scheduling in Dynamic Urban Safety"
   Authors: Majid Ghasemi, Ibrahim Sorkhoh, Fadi Alzhouri, and Dariush Ebrahimi
   Status: Submitted to *IEEE Industrial Informatics: Cloud Computing, Big Data, AI, Informatics and Software Engineering*

3. **Title:** "Comparative Analysis of Reinforcement Learning for Reliable Real-Time EV Routing and Charging"
   Authors: Majid Ghasemi, Dariush Ebrahimi, and Fadi Alzhouri
   Status: Accepted to *2025 21st International Conference on the Design of Reliable Communication Networks (DRCN)*

Preprints on arxiv

1. **Title:** "A Comprehensive Survey of Reinforcement Learning: From Algorithms to Practical Challenges"
   Authors: Majid Ghasemi, Amir Hossein Moosavi, and Dariush Ebrahimi
   Status: Available on arxiv.org

2. **Title:** "Introduction to Reinforcement Learning: Key Concepts, Algorithms, and Categories"
   Authors: Majid Ghasemi and Dariush Ebrahimi
   Status: Available on arxiv.org

Table of Contents

Examining Committee ii

Author's Declaration iii

Abstract iv

Acknowledgements vi

Publications vii

List of Figures xiii

List of Tables xvi

1 Introduction 1

2 Foundations of Reinforcement Learning 4

2.1 Introduction 4

2.2 Tabular Model-based Algorithms 7

2.2.1 Dynamic Programming (DP) 8

2.2.2 Policy Iteration 8

2.2.3 Value Iteration 9

2.3 Tabular Model-free Algorithms 10

2.3.1 Monte Carlo (MC) Methods 10
2.3.2 Monte Carlo Estimation of State Values 11
2.3.3 MC Estimation of Action Values (with Exploration Starts) 13
2.3.4 On-policy MC (MC without Exploration Starts) 15
2.3.5 Off-policy Prediction via Importance Sampling 16
2.3.6 MC with Importance Sampling 19
2.3.7 Temporal Difference (TD) Learning 21
2.3.8 TD(0) 21
2.3.9 TD(0)-Replay 22
2.3.10 TD(λ) 23
2.3.11 N-step Bootstrapping 26
2.3.12 N-step TD Prediction 26
2.3.13 N-step Off-policy Learning 29
2.3.14 Q-learning 31
2.3.15 Double Q-learning 35
2.3.16 State-Action-Reward-State-Action (SARSA) 39
2.3.17 Expected SARSA 43
2.3.18 N-step SARSA 45
2.4 Analytical Experiments 48
2.5 Conclusion 54

3 Deep Reinforcement Learning **55**
3.1 Introduction 55
3.1.1 Deep Learning Overview 56
3.2 Approximation Value-based Methods 56
3.2.1 Deep Q-Networks (DQN) 57
3.2.2 Double Deep Q-Networks (DDQN) 62
3.2.3 Dueling Deep Q-Networks (Dueling DQN) 68

3.2.4 Advanced Tabular Model-based Methods 73
3.2.5 Model-based Planning 73
3.3 Policy-based Methods 98
3.3.1 REINFORCE 99
3.3.2 Trust Region Policy Optimization (TRPO) 104
3.3.3 Proximal Policy Optimization (PPO) 111
3.4 Actor-Critic Methods 118
3.4.1 A3C & A2C 118
3.4.2 Deterministic Policy Gradient (DPG) 128
3.4.3 Deep Deterministic Policy Gradient (DDPG) 129
3.4.4 Twin Delayed Deep Deterministic Policy Gradient (TD3) 135
3.5 Analytical Experiments 140
3.6 Conclusion 143

4 Electric Vehicles Routing & Charging 144
4.1 Introduction 144
4.2 System Model and Problem Description 146
4.3 The Heuristic Approach: Dynamic TERC2 147
4.4 Reinforcement Learning Approaches 149
4.4.1 Q-Learning 151
4.4.2 TQ-Learning (TQL) 152
4.4.3 Deep Q-Learning (Deep Q-Networks) 153
4.5 Performance Evaluation 155
4.5.1 Graph Generation & Charging Station Placement 156
4.5.2 Performance Evaluation over Different Networks 159
4.6 Conclusion 160

5 Vehicle Patrol Scheduling **163**
5.1 Introduction . . . 163
5.2 Related Work . . . 165
5.2.1 Optimization and Heuristic-Based Approaches . . . 165
5.2.2 RL-Based Approaches . . . 168
5.3 Optimization, Heuristic, and Genetic Approaches . . . 170
5.3.1 System Model and Problem Definition . . . 170
5.3.2 Problem Formulation . . . 173
5.3.3 Adaptive Hill-Climbing-Based Patrol Scheduling (AHBPS) . . . 177
5.3.4 Genetic-Based Dynamic Vehicle Patrol Scheduling (GDVPS) . . . 181
5.3.5 Performance Evaluation . . . 183
5.4 Reinforcement Learning & Heuristic Approach for VPS with Emergency Cases 193
5.4.1 System Model and Problem Definition . . . 194
5.4.2 The Heuristic Method: AHBPS2 . . . 198
5.4.3 The Reinforcement Learning Approach: $\mathcal{P}^4\mathcal{O}$ . . . 201
5.4.4 Performance Evaluation . . . 205
5.5 Conclusion . . . 214
5.6 Appendix . . . 214

6 Conclusion and Future Work **217**

References **219**

APPENDICES **248**

List of Figures

2.1 Agent-environment interaction in RL [1] 5
2.2 Maze Grid Environment 48
2.3 Task 1 Comparison 50
2.4 Task 2 Comparison 51
2.5 Task 3 Comparison 53

3.1 The architecture of DQN 58
3.2 The basic MCTS process [2] 74
3.3 One iteration of the general MCTS approach 75
3.4 L_{CLIP} for one timstep in PPO 112
3.5 The A3C architecture 120
3.6 Farama Highway environment 141

4.1 EV routing and charging system framework 147
4.2 Comparing the traveled distance (in KM) of different methods (TQL, Q-Learning, and DQL) for (a) 125 nodes, (b) 250 nodes, and (c) 500 nodes. . 156
4.3 Comparing the traveled time (in Minutes) of different methods (TQL, Q-Learning, and DQL): (a) 125 nodes, (b): 250 nodes, and (c): 500 nodes . . 157
4.4 Comparing the achieved cumulative reward of different methods (TQL, Q-Learning, and DQL): (a) 125 nodes, (b): 250 nodes, and (c): 500 nodes . . 158
4.5 Comparing the traveled distance of different methods (TQL, Q-Learning, and DQL) in a network of 125 nodes: (a) radius = 30, (b) radius = 45, and (c) radius = 60 (baseline). 162

5.1 VPS System Model 171
5.2 GDVPS Flowchart 182
5.3 Waterloo Map 184
5.4 Extracted Waterloo's graph 185
5.5 Performance comparison (total visits) of OP, AHBPS, and GDVPS methods under different scenarios 186
5.6 Execution time comparison between OP, AHBPS and GDVPS 187
5.7 Total visits comparison of AHBPS and GDVPS methods under different scenarios 188
5.8 Execution time comparison between AHBPS and GDVPS 189
5.9 Total traveled distance (in kilometers) comparison between AHBPS and GDVPS 190
5.10 Total visits comparison between the proposed methods with different patrol time 192
5.11 Total visits comparison between the proposed methods with different rest period 192
5.12 Performance comparison (total visits) between the proposed methods with different travel time distributions (standard deviation is fixed to 5) 194
5.13 Total visits comparison between the proposed methods with different operation time 195
5.14 $\mathcal{P}^4\mathcal{O}$ System Figure 197
5.15 Total visits comparison of $\mathcal{P}^4\mathcal{O}$ & AHBPS2 methods under the mentioned scenarios 206
5.16 Training progress for $\mathcal{P}^4\mathcal{O}$ 207
5.17 Comparison of the proposed methods regarding the total traveled distance (KM) 208
5.18 Total visits comparison for the proposed methods with different Patrol Time 209
5.19 Total visits comparison for the proposed methods with different Rest Time 210
5.20 Performance comparison (total visits) between the proposed methods with different travel time distributions (standard deviation is fixed to 5) 211

5.21 Total visits comparison for the proposed methods with different Operation Time 211
5.22 Emergency handling success rate comparison for the proposed methods . . 212
5.23 Emergency handling success rate comparison for the proposed methods with different emergency rates 213

List of Tables

5.1 Notations used for the problem formulation 174
5.2 Notations used in AHBPS 177
5.3 Parameters and their Values used in our simulations 185
5.4 Notation Definitions 196
5.5 $\mathcal{P}^4\mathcal{O}$ Simulation Parameters. 205

6.1 A list of notations and symbols used in Chapter 2 250
6.2 A list of notations and symbols used in Chapter 3 251
6.3 A complete list of notations and symbols used in Chapter 4 252
6.4 A complete list of notations and symbols used in Chapter 5 253
6.5 List of Abbreviations Used in This Thesis (Part 1) 255
6.6 List of Abbreviations Used in This Thesis (Part 2) 256

Chapter 1

Introduction

Reinforcement Learning (RL) has become a prominent framework for sequential decision-making in settings characterized by uncertainty, complexity, and dynamic interactions. By iteratively refining an agent's policy through trial and error, RL bridges the gap between theoretical learning models and real-world applications, including robotic control, resource scheduling, and adaptive routing. However, as RL methods evolve, particularly with the incorporation of deep function approximators, they face emerging challenges related to stability, scalability, and sample efficiency. This thesis addresses these theoretical and practical issues, demonstrating how traditional RL concepts can be expanded and adapted to handle various tasks.

The central motivation for this research stems from the increasing demand for data-driven decision-making in domains where environments and constraints are subject to rapid changes. Conventional planning and optimization approaches often prove inadequate in nonstationary or high-dimensional contexts. By allowing an agent to learn optimal or near optimal actions directly from interactions, RL mitigates the need for complete environmental models. In so doing, RL provides a robust paradigm for problems ranging from transportation and logistics to control systems and beyond. A primary goal of this thesis is to investigate how fundamental RL ideas, focused on both model-based and model-free methodologies, can be integrated with modern computational techniques to address theoretical limitations and real-world performance hurdles. In pursuit of this goal, the research seeks to answer the following questions:

- **Dynamic Adaptation:** How can reinforcement learning techniques be adapted to effectively solve dynamic scheduling and routing problems in real-world environments?

- **Algorithmic Performance:** What are the critical factors that influence the performance, convergence, and stability of RL algorithms when applied to electric vehicle charging and routing as well as vehicle patrol scheduling?
- **Hybrid Approaches:** How can integrating domain-specific heuristics with classical and deep RL methods enhance real-time decision-making and scalability in complex routing and scheduling scenarios?

To explore these themes, the thesis is organized into five chapters. This introductory chapter presents the overarching research context, objectives, and scope. Chapter 2 outlines the foundations of RL, examining Markov Decision Processes and core solution methods built upon them, including Dynamic Programming, Monte Carlo, and Temporal Difference algorithms. Controlled experiments illustrate how variations in exploration strategies, discount factors, or eligibility traces can influence both convergence properties and learning speed.

Chapter 3 shifts the focus to deep RL methods, highlighting the role of neural networks in approximating value functions and policies for high-dimensional problems. It discusses Deep Q Networks (DQN), Double DQN, and policy gradient approaches such as Proximal Policy Optimization (PPO) and Asynchronous Advantage Actor Critic (A3C). Empirical results in more complex environments illustrate how these methods can surpass classical tabular algorithms, although they may involve trade-offs in sample efficiency and sensitivity to hyperparameters.

In Chapter 4, the electric vehicle routing and charging problem is addressed, requiring the identification of efficient paths while accounting for battery constraints, limited charging infrastructure, and evolving traffic conditions. Three reinforcement learning approaches are assessed: tabular Q-learning (QL), a heuristic-augmented Q-learning method (TQL), and a deep Q-learning strategy (DQL). Although QL converges under smaller networks, it struggles to handle the rapidly expanding state space caused by traffic variability and fine-grained battery levels. TQL partially overcomes these issues through rules that guide charging decisions and reduce search complexity, yet it retains a discrete table that becomes unwieldy as the network grows. DQL instead employs a neural network to represent the action-value function, enabling generalization across high-dimensional states and making effective use of limited experiences. Results indicate that DQL maintains higher total rewards and achieves lower travel distances and fewer recharging stops than either QL or TQL, particularly when traffic fluctuations increase and the network doubles in size. Its learned approximations adapt more robustly to novel road conditions, suggesting that deep function approximation not only streamlines learning but also scales to larger, more realistic scenarios that demand dynamic, time-sensitive routing decisions.

Chapter 5 addresses vehicle patrol scheduling where multiple patrol units must repeatedly visit a set of critical checkpoints subject to constraints such as minimum revisit frequencies, rest periods, and unplanned emergencies. Various solution strategies are developed and compared. First, a classical optimization method, formulated as a mixed integer linear program, is implemented to derive theoretically optimal patrol routes given static conditions; however, this approach becomes computationally expensive and less adaptable as the problem size grows or as real-time changes occur. Next, two heuristic-based solutions are presented: an adaptive hill-climbing algorithm that incrementally refines a candidate solution and a genetic algorithm that evolves a population of patrol schedules using crossover and mutation operators. Both heuristics are more computationally efficient than the pure optimization model, yet they struggle to dynamically reassign vehicles when a sudden surge in emergency calls renders their schedules suboptimal. To handle volatile environments with high uncertainty, an RL framework is proposed. In particular, the RL agent exploits historical data and near-real-time feedback to update its policy, enabling it to adapt routes on the fly as new emergency incidents arise or traffic times fluctuate. The agent's state includes the vehicles' locations, battery or fuel states if applicable, upcoming rest demands, and the distribution of newly reported incidents; its action space encompasses dispatching vehicles to the next checkpoint or emergency site. Because the agent refines its policy incrementally, it learns which routes optimize both regular patrol coverage and emergency responsiveness. Simulation results show that the RL-based approach achieves broader patrol coverage, lower average response times, and more balanced vehicle workloads than either the classical optimization or heuristic-based alternatives. The capacity to incorporate updates online allows the RL agent to scale to larger networks, where static or batch-oriented methods incur heavy computational costs or fail to adapt to sudden changes. These findings underscore the effectiveness of combining domain heuristics with data-driven RL policies for continuous fleet management tasks, especially under significant uncertainty and rapid variations in demand and traffic conditions.

The thesis concludes by synthesizing the contributions and highlighting their broader significance for RL research and practice. Overall, this work emphasizes the versatility of RL and the importance of approaches that combine theoretical rigor with practical adaptability to address large-scale decision-making problems.

Chapter 2

Foundations of Reinforcement Learning

In this chapter, we begin by exploring traditional reinforcement learning (RL), reviewing key algorithms and foundational research from dynamic programming to tabular model-free methods. Finally, we evaluate selected algorithms on a grid-world maze problem.

2.1 Introduction

In the framework of RL, the learning process is defined by several key components. The fundamental concept of RL is to capture the crucial elements of a real problem faced by an **agent** that interacts with its **environment** to achieve a goal (Fig 2.1). It is evident that such an agent must be capable of sensing the state of the environment to some extent and must have the ability to take actions that influence the state [3]. In general, an **action** refers to any decision an agent will have to make, while a **state** refers to any factor that the agent will have to consider when making that decision. Beyond the agent and the environment, an RL system has four main sub-elements: A policy, a reward signal, a value function, and, optionally, a model of the environment. The **reward signal** determines the agent's behavior in the given environment. During each time step, the environment sends a scalar signal, a reward, to the agent. Ultimately, the agent's objective is to maximize the total reward it receives. A **policy** defines how a learning agent should behave at a particular point in time. In simple terms, the agent's policy is the mapping from a set of possible states to actions. The **value function** corresponds to the agent's current mapping from the set of possible states to its estimates of the net long-term reward it can expect

once it visits a state (or state-action pair) and continues to follow its current policy. As a final point, the **model of the environment** simulates the behavior of the environment or, more generally, can be used to infer how the environment will behave [3, 4, 5].

Figure 2.1: Agent-environment interaction in RL [1]

There are two broad categories of RL methodology: Model-free and model-based. Model-free methods do not assume knowledge of the environment's dynamics and learn directly from interactions with the environment. On the other hand, model-based methods involve building a model of the environment's dynamics and using this model to develop and improve policies [6, 1].

Following this, we briefly examine two of the most crucial components of RL. First, we explore the Markov Decision Process (MDP), the foundational framework that structures the learning environment and guides the agent's decision-making process. Then, we discuss the exploration-exploitation dilemma, one of the most imperative characteristics of RL, which balances the need to gather new information to maximize rewards [3].

The MDP is a sequential decision-making process in which the costs and transition functions are directly related to the current state and actions of the system. MDPs aim to provide the decision-maker with an optimal policy.

MDP is defined by a tuple (S, A, P, R, γ), where:

- S is a finite set of states,

- A is a finite set of actions,
- $P : S \times A \times S \to [0, 1]$ is the state transition probability function, where $P(s'|s, a)$ denotes the probability of transitioning to state s' from state s by taking action a,
- $R : S \times A \to \mathbb{R}$ is the reward function, where $R(s, a)$ denotes the immediate reward received after taking action a in state s,
- $\gamma \in [0, 1]$ is the discount factor that determines the importance of future rewards.

The agent's behavior is defined by a policy $\pi : S \to A$, which maps states to actions. The goal of the agent is to find an optimal policy π^* that maximizes the expected cumulative reward, often termed the return (G_t), which is defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}) \tag{2.1}$$

Central to solving an MDP are value functions, which estimate the expected return. The state-value function $V_\pi(s)$ under policy π is the expected return starting from state s and following policy π thereafter [1, 7, 8]:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] \tag{2.2}$$

Similarly, the action-value function $Q_\pi(s, a)$ is the expected return starting from state s, taking action a, and thereafter following policy π:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] \tag{2.3}$$

Equations 2.2 and 2.3 are referred to **Bellman equations**. To find the optimal policy π^*, RL algorithms iteratively update value functions based on experience [9, 10]. In Q-learning, the update rule for the action-value function is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(R(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\Big) \tag{2.4}$$

Where α is the learning rate and γ is the discount factor. For detailed explanations of MDPs and RL in general, readers are referred to [3, 6, 11, 12, 13, 14, 1]

It is important to note that exploration and exploitation represent a fundamental trade-off in RL. The objective of exploration is to discover the effects of new actions, whereas the

objective of exploitation is to select actions that have been shown to yield high rewards, to the best of the knowledge of the agent at that timestep. Exploration techniques can be classified into two: Undirected [15] and directed [16, 17] exploration methods. Undirected exploration methods such as semi-uniform (ϵ-greedy) exploration [18, 19, 20] and the Boltzmann exploration [21, 22, 23] try to explore the whole state-action space by assigning positive probabilities to all possible actions. On the other hand, directed exploration methods like the E^3 algorithm [24] and exploration bonus [25] use the statistics obtained through past experiences to execute efficient exploration.

Balancing exploration and exploitation is a critical aspect of RL, and various strategies have been developed to manage this trade-off effectively. For instance, in [26], authors introduced a model-based RL method that dynamically balances exploitation and exploration, particularly in changing environments. By using Bayesian inference with a forgetting effect, it estimates state-transition probabilities and adjusts the balance parameter based on action-outcome variations and environmental changes. Furthermore, authors in [27] proposed Decoupled RL (DeRL), which trains separate policies for exploration and exploitation and can be applied with on-policy and off-policy RL algorithms. Using decoupling policies, DeRL improves robustness and sample efficiency in sparse reward environments. More advanced methodologies have been used in [28], where the study investigated the trade-off between exploration and exploitation in continuous-time RL using an entropy-regularized reward function. As a result, the optimal exploration-exploitation balance was achieved through a Gaussian distribution for the control policy, where exploitation was captured by the mean and exploration by the variance. Moreover, various strategies such as Upper Confidence Bound (UCB) [3] and Thompson sampling [29] are employed to balance this trade-off in simpler environments, such as bandits.

In the next section, we dive into the first group of RL algorithms, tabular model-based methods.

2.2 Tabular Model-based Algorithms

In this section, we examine the first part of tabular model-based methods in RL, specifically DP methods. A tabular model-based algorithm is an example of an RL technique used for solving problems with a finite and discrete state and action space. These algorithms are explicitly based on the maintenance and updating of a table or matrix, which represents the dynamic nature of the environment and its reward. Model-based algorithms are characterized by the fact that they involve the construction of an environmental model for use in decision-making. Key characteristics of tabular model-based techniques include

model representation, planning and policy evaluation, and value iteration [3]. In model representation, the model depicts the dynamics in a tabular form with transition probabilities represented as $P(s'|s,a)$ and the reward function as $R(s,a)$. These elements define the probability of transitioning to a new state s' and the expected reward when taking action a in state s. Planning and policy evaluation involves approximating the updating value function $V(s)$ iteratively using the Bellman equation until convergence. To calculate the optimal policy, the algorithm examines all possible future states and their associated actions. Value iteration also approximates the updating value function $V(s)$ iteratively using the Bellman equation until convergence. Similar to planning and policy evaluation, it analyzes all potential future states and their linked actions to determine the optimal policy. Tabular model-based algorithms can be divided into two general categories: DP and model-based planning, where this variant is discussed in our next survey.

2.2.1 Dynamic Programming (DP)

DP methods are fundamental techniques used to solve MDPs when a complete model of the environment is known. These methods are iterative and make use of the Bellman equations to compute the optimal policies. Two primary DP methods are **policy iteration** and **value iteration**.

2.2.2 Policy Iteration

Policy iteration is a method that iteratively improves the policy until it converges to the optimal policy. It consists of two main steps: Policy evaluation and policy improvement [3]. **Policy evaluation** calculates the value function $V_\pi(s)$ for a given policy π. This involves solving the Bellman expectation equation for the current policy:

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_\pi(s')] \tag{2.5}$$

This step iteratively updates the value of each state under the current policy until the values converge [1]. **Policy improvement**, on the other hand, improves the policy by making it greedy with respect to the current value function:

$$\pi'(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_\pi(s')] \tag{2.6}$$

Algorithm 1 Policy Iteration

```
Initialize policy π arbitrarily
repeat
    Perform policy evaluation to update the value function V_π
    Perform policy improvement to update the policy π
until policy π converges
```

This step updates the policy by selecting actions that maximize the expected value based on the current value function [30]. Alg. 1 gives an overview of how policy iteration can be implemented.

2.2.3 Value Iteration

Value iteration is another DP method that directly computes the optimal value function by iteratively updating the value of each state. It combines the steps of policy evaluation and policy improvement in a single step. Value iteration consists of two different steps: value update and policy extraction. **Value update** revises the value function for each state based on the Bellman optimality equation:

$$V(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')] \tag{2.7}$$

This step involves iterating through all states and updating their values based on the maximum expected return of the possible actions. **Policy extraction** happens once the value function has converged; the optimal policy can be extracted by selecting actions that maximize the expected value:

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')] \tag{2.8}$$

Alg. 2 illustrates the implementation of value iteration. Value iteration is simpler to implement than policy iteration since it combines evaluation and improvement into a single process. Additionally, it is often faster in practice for many problems, as it does not require separate policy evaluation steps [31]. On the downside, value iteration may require a large number of iterations to converge, especially for problems with large state spaces. Furthermore, it can be less stable than policy iteration in some cases due to the combined update steps [3].

Algorithm 2 Value Iteration

```
Initialize the value function V(s) arbitrarily
repeat
    for each state s do
        Update V(s) using the Bellman optimality equation
    end for
until the value function V(s) converges
Extract the optimal policy π* from the converged value function
```

Throughout the next section, we start exploring tabular model-free algorithms by introducing MC methods first.

2.3 Tabular Model-free Algorithms

Tabular model-free algorithms are techniques suitable for problems with discrete state and action spaces that are typically small enough to be tabulated. Unlike model-based algorithms, which require a model of the environment's dynamics, model-free algorithms learn directly from interactions with the environment without understanding its underlying mechanics. In this context, we explore various methods and algorithms that are categorized as tabular model-free algorithms, starting with the MC Methods.

2.3.1 Monte Carlo (MC) Methods

When complete knowledge of the environment is either unavailable or impractical, MC methods provide an effective alternative. These methods learn from experience by utilizing sampled sequences of states, actions, and rewards obtained through real or simulated interactions with the environment [3]. A key advantage of learning from real experience is that MC does not require prior knowledge of the environment's dynamics while still being capable of achieving optimal behavior. Similarly, learning from simulated experience is advantageous, as it allows for effective policy learning with a model that only needs to generate sample transitions rather than the full probability distributions of all possible transitions, as required in DP. To ensure well-defined returns, MC methods are primarily applied to episodic tasks, where experiences are divided into episodes that always reach a terminal state regardless of the chosen actions. Value estimates and policies are updated only at the end of an episode based on sample returns. Consequently, MC methods operate

incrementally on an episode-by-episode basis rather than in a step-by-step (online) manner. Although the term "Monte Carlo" is often broadly associated with estimation techniques involving significant randomness, in this context, it specifically refers to methods that compute averages over complete returns [3].

Next, we describe how MC methods are utilized to learn the state-value function for a given policy π, outlining the various approaches available for this purpose.

2.3.2 Monte Carlo Estimation of State Values

A fundamental approach to estimating the value of a state from experience is to compute the average of its observed returns after each visit. The value of a state is defined as the expected cumulative discounted reward in the future. By the law of large numbers, this average converges to the expected value as more returns are observed, forming the foundation of MC methods. The value function $v_\pi(s)$ for a state s under policy π is estimated using a set of episodes generated by following π and passing through s. Each occurrence of s within an episode is referred to as a visit. If s appears multiple times in a single episode, the first occurrence is termed the first visit. The **first-visit MC method** estimates $v_\pi(s)$ by averaging returns only from first visits to the state, whereas the **every-visit MC method** averages returns from all visits to s.

To illustrate, consider a simple 3×3 grid world where an agent begins at the **top-left corner** (state S_0) and aims to reach the **bottom-right corner** (goal state G). The agent can choose from four actions in each state: up, down, left, or right. Each action moves the agent to an adjacent state unless it would result in leaving the grid, in which case the agent remains in its current state. The episode concludes when the agent reaches the goal state G.

Key Rules:

1. States can be revisited during an episode.
2. The return at the goal state is defined as the sum of rewards from the starting state to the goal state.
3. Two approaches are considered for state-value estimation:
   - **First-visit MC:** Only the first visit to each state in an episode is used for value estimation.
   - **Every-visit MC:** All visits to each state in an episode are considered.

Agent's Behavior Across Episodes:

Episode 1:

- **Actions:** Right, Right, Down, Down (reaches G).
- **Returns:** G is visited once with a return of 5.

Episode 2:

- **Actions:** Up, Right, Right, Down, Down (reaches G).
- **Returns:** G is visited twice, with returns of 8 (first visit) and 8 (second visit).

Episode 3:

- **Actions:** Right, Up, Right, Down, Down (reaches G).
- **Returns:** G is visited once with a return of 6.

State-value Estimation for G:

1. **First-visit MC:**
   - Considers only the **first visit** to G in each episode.
   - Returns for G:
     - Episode 1: 5
     - Episode 2: 8 (first visit only)
     - Episode 3: 6
   - **Average return for G:** $\frac{5+8+6}{3} = 6.33$.
2. **Every-visit MC:**
   - Considers **all visits** to G in each episode.
   - Returns for G:
     - Episode 1: 5
     - Episode 2: $8, 8$
     - Episode 3: 6

Algorithm 3 First-visit MC [3]

1: Initialize:
2: $\pi \leftarrow$ policy to be evaluated
3: $V \leftarrow$ an arbitrary state-value function
4: Returns(s) $\leftarrow$ an empty list, for all $s \in \mathcal{S}$
5: **repeat**
6: Generate an episode using π
7: **for** each state s appearing in the episode **do**
8: $G \leftarrow$ return following the first occurrence of s
9: Append G to Returns(s)
10: $V(s) \leftarrow$ average(Returns(s))
11: **end for**
12: **until** forever

- **Average return for G:** $\frac{5+8+8+6}{4} = 6.75$.

General Insights:

- Using **first-visit MC**, each state value is updated using the average return from its first visits across episodes (e.g., G in Episode 2 considers only the first return, 8).
- Using **every-visit MC**, the state value reflects the average of all returns across all visits to that state.

In MC, both the first-visit and every-visit methods converge to $v_\pi(s)$ as the number of visits approaches infinity [3]. Algorithm 3 outlines the first-visit MC approach for estimating V_π. The only distinction between the two methods, as noted earlier, is in line 8, where every-visit MC updates the value function based on all occurrences of state s rather than only the first visit.

2.3.3 MC Estimation of Action Values (with Exploration Starts)

In the absence of an environment model, estimating action values (values associated with state-action pairs) rather than state values becomes particularly advantageous. While state values are sufficient for deriving a policy by evaluating one step ahead and selecting the action that maximizes the expected reward and next-state value [3], they alone are inadequate without a model. An explicit estimate of each action's value is necessary to

provide meaningful policy guidance. A deterministic policy π poses a major challenge by restricting exploration to a limited subset of state-action pairs, as only one action is observed per state. Consequently, MC estimates for unvisited actions do not improve with experience, leading to a critical learning limitation. By estimating action values, an agent can evaluate all available actions in each state, facilitating more informed decision-making. It is essential to estimate the value of all actions from each state rather than only the currently favored one to ensure meaningful comparisons.

While exploring starts can be useful in some cases, they are not always feasible, particularly when learning directly from the environment, where initial conditions may not be favorable. As an alternative, stochastic policies that assign a non-zero probability to all actions in each state can ensure that all state-action pairs are sufficiently explored. To introduce an MC-based adaptation of classical policy iteration, we begin by iteratively evaluating and improving policies. This approach starts with an arbitrary policy π_0 and progressively refines it until both the optimal policy and the optimal action-value function are obtained.

$$\begin{aligned} &\pi_0 \xrightarrow{\text{Evaluation}} q_{\pi_0} \xrightarrow{\text{Improvement}} \pi_1 \xrightarrow{\text{Evaluation}} q_{\pi_1} \\ \xrightarrow{\text{Improvement}} \pi_2 \xrightarrow{\text{Evaluation}} &\dots \xrightarrow{\text{Improvement}} \pi_* \xrightarrow{\text{Improvement}} q_{\pi_*} \end{aligned} \tag{2.9}$$

As the approximate action-value function asymptotically converges to the true function, numerous episodes occur. For now, we assume an infinite number of observed episodes, all generated with exploration starts. Exploration starts refer to the assumption that all states have a nonzero probability of being the starting state. While this assumption promotes exploration, it is not always practical, as it does not hold in many real-world applications. Under these assumptions, for any arbitrary policy π_k, MC methods can accurately estimate q_{π_k}.

To improve the policy, it must be made greedy with respect to the current value function. This is achieved by utilizing the action-value function to construct a greedy policy without requiring a model. Given an action-value function q, a greedy policy selects the action that maximizes the action-value for each state $s \in S$.

$$\pi(s) \equiv \arg\max_a q(s, a) \tag{2.10}$$

Policy improvement is achieved by defining each π_{k+1} as the greedy policy with respect to q_{π_k}. The Policy Improvement Theorem, previously discussed, applies to π_k and π_{k+1}, since for all $s \in S$, the following holds:

$$q_{\pi_k}(s, \pi_{k+1}(s)) = q_{\pi_k}(s, \arg\max_a q_{\pi_k}(s, a)) = \max_a q_{\pi_k}(s, a) \geq q_{\pi_k}(s, \pi_k(s)) = v_{\pi_k}(s) \quad (2.11)$$

According to the General Policy Theorem [3], each π_{k+1} is either strictly superior to or equally optimal as π_k, ensuring that both policies converge to the optimal solution. Consequently, this process leads to an optimal value function and policy. Using MC methods, optimal policies can be derived exclusively from sample episodes without requiring explicit knowledge of the environment's dynamics.

Although MC simulations guarantee convergence, two key assumptions must be addressed for practical implementation: the requirement of exploration starts in episodes and the necessity of an infinite number of episodes for policy evaluation. Our focus is on eliminating the assumption of infinite episodes. This can be accomplished by approximating q_{π_k} during each evaluation, leveraging empirical measurements and assumptions to minimize error. While this approach theoretically guarantees correct convergence, it often necessitates an impractically large number of episodes, particularly in complex environments. An alternative is to avoid full policy evaluation before updating the policy. Instead, the value function is incrementally adjusted toward q_{π_k} over multiple steps, as observed in generalized policy iteration. A similar approach is seen in value iteration, where policy improvement occurs after just one policy evaluation step.

MC policy iteration inherently alternates between evaluation and improvement at the end of each episode. The returns observed within an episode are utilized for evaluation, followed by policy improvement for all visited states. While the assumption of exploration simplifies estimation by ensuring that all state-action pairs are encountered, it is often impractical. By relaxing this assumption, an on-policy MC approach can be employed, where action values are estimated directly from trajectories generated by the policy itself. In the following discussion, we examine this alternative approach, which does not rely on exploration starts.

2.3.4 On-policy MC (MC without Exploration Starts)

MC with on-policy starts, also referred to as MC without exploration starts, is a learning approach where the agent derives knowledge from episodes generated by following its current policy. Unlike methods that employ exploratory starts, allowing the agent to begin from any state-action pair, on-policy MC relies solely on actual experiences encountered while executing the policy in the environment. Consequently, the agent estimates the value

of the policy by averaging cumulative rewards from episodes that begin and terminate according to the policy. Ensuring sufficient exploration is crucial for obtaining accurate value estimates, as the agent's experiences are restricted to those dictated by its current policy. This can be achieved by employing stochastic policies or incorporating soft exploration strategies. The approach is particularly useful in scenarios where exploration starts are impractical or where the policy must be learned through direct environmental interaction [3, 32].

We highlight two studies in this section, recognizing that numerous other contributions exist in the literature.

In [33], researchers introduced an off-policy MC learning method that incorporated approximation spaces and rough set theory. This approach refined the RL process in dynamic environments by employing weighted sampling based on observed behavior patterns, enabling a more precise estimation of action values and enhancing learning efficiency and adaptability. The method's success depended on accurately identifying and leveraging behavior patterns, particularly in complex environments. A more extensive comparison with other advanced RL techniques could have further clarified its effectiveness and scalability across various applications.

In [34], the authors developed an approach that advanced RL by utilizing rough set theory and approximation spaces to evaluate state values within MC without exploration starts. Tested in a simulation modeling zebra danio fish behavior, this method integrated observed patterns to refine action-value estimations, demonstrating the potential for modeling complex systems. While the technique was innovative, it introduced computational challenges, underscoring the trade-off between methodological advancement and practical feasibility. The research highlighted the potential of pattern-based evaluation while emphasizing the importance of scalability and broader implementation.

In the next subsection, we analyze the integration of importance sampling [35], a well-known concept in statistics, with MC methods.

2.3.5 Off-policy Prediction via Importance Sampling

The use of RL for control presents a fundamental challenge: the need to learn action values based on the assumption of subsequent optimal behavior while also requiring non-optimal behavior to explore all possible actions in search of the optimal one. This raises the question of how to learn optimal policies while following exploratory policies. A policy-based approach provides a compromise by learning action values for a policy that, while not necessarily optimal, remains close to optimal and includes exploration mechanisms.

However, due to its exploratory nature, this approach does not directly resolve the problem of learning the optimal action values.

Off-policy learning addresses this challenge by utilizing two distinct policies: a target policy (π), which aims to become the optimal policy, and a behavior policy (b), which generates behavior. This dual-policy framework enables exploration to be conducted separately from the process of learning the optimal policy, allowing the agent to learn from data generated outside the target policy. Off-policy methods are more flexible and powerful than on-policy approaches, with on-policy methods being a special case where the target and behavior policies are identical. However, off-policy methods introduce additional complexity, necessitating more advanced concepts and notation. Since off-policy learning relies on data from a different policy, it tends to have higher variance and slower convergence compared to on-policy learning. While on-policy methods offer simplicity and allow direct learning from the agent's own exploratory actions, off-policy methods provide a more robust framework for learning optimal policies indirectly through a separate behavior policy. Ultimately, the distinction between on-policy and off-policy learning reflects the exploration-exploitation trade-off inherent in RL, enabling agents to effectively learn and adapt in complex environments [1]. Alg. 4 provides a general overview of this algorithm.

In the following paragraphs, we examine selected research studies that have applied MC methods and their variations, evaluating their underlying rationale and addressing the challenges they encountered. By conducting an in-depth analysis of these studies, we aim to highlight the versatility and effectiveness of MC methods in solving diverse problems while emphasizing the decision-making processes that led to their adoption.

Using MC with historical data, [36] proposed an intelligent train control approach aimed at enhancing energy efficiency and punctuality. This model-free method achieved a 6.31% reduction in energy consumption while improving schedule adherence. However, its effectiveness was highly dependent on the availability and quality of historical data, and its scalability to larger, more complex railway networks remained an open question.

In [37], the authors introduced a Renewal MC (RMC) algorithm designed to reduce variance, minimize update delays, and accelerate convergence to locally optimal policies. The approach demonstrated strong performance in continuous state and action spaces and was applicable to various fields, including robotics and game theory. Additionally, an approximate version was proposed to improve convergence speed while maintaining bounded errors. However, the performance of RMC was influenced by the choice and size of the renewal set.

The study by [34] introduced an MC off-policy strategy enhanced with rough set theory, presenting a novel methodology in the field. While this integration provided new insights,

Algorithm 4 Off-policy Prediction (via Importance Sampling) [3]

```
Initialize, for all s ∈ S, a ∈ A(s):
    Q(s, a) ∈ ℝ (arbitrarily)
    C(s, a) ← 0
    π(s) ← arg max_a Q(s, a) ( ties broken)
repeat
    b ← any soft policy
    Generate an episode using b: (S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T)
    G ← 0
    W ← 1
    for each step of episode, t = T − 1, ..., 0: do
        G ← γG + R_{t+1}
        C(S_t, A_t) ← C(S_t, A_t) + W
        Q(S_t, A_t) ← Q(S_t, A_t) + W/C(S_t, A_t) [G − Q(S_t, A_t)]
        π(S_t) ← arg max_a Q(S_t, a)(with ties broken consistently)
        if A_t ≠ π(S_t) then
            break                                   ▷ Proceed to next episode
        end if
        W ← W · 1/b(A_t|S_t)
    end for
until forever
```

its complexity posed challenges for broader applicability. Furthermore, the research primarily focused on theoretical formulations, requiring additional empirical validation.

Authors in [38] developed a Bayesian model-free Markov chain MC algorithm for policy search, applied specifically to a 2-DoF robotic manipulator. The algorithm proved to be both practical and effective in real-world implementations, utilizing a gradient-free strategy that simplified optimization and excelled in controlling complex trajectories with relatively few iterations. However, its applicability might be restricted to specific domains, and the estimator's high variance introduced challenges.

The work by [39] introduced an innovative MC Bayesian RL approach that streamlined Bayesian RL (BRL). By sampling a limited set of hypotheses, the method constructed a discrete Partially Observable Markov Decision Process (POMDP), eliminating the reliance on conjugate distributions and enabling the use of point-based approximation algorithms. This framework was adaptable to both fully and partially observable environments, demonstrating robust performance across various domains. However, its effectiveness was con-

tingent on the choice of prior distributions, and insufficient sample sizes could degrade performance.

In [40], the authors proposed a factored MC BRL method to address BRL problems in an online setting. By leveraging factored representations, the approach reduced the complexity of learning parameters and employed partially observable MC planning as an online solver. This significantly enhanced scalability and efficiency, particularly in large-scale domains.

Researchers in [41] explored the development of self-learning agents for the Batak card game using MC methods for state-value estimation combined with Artificial Neural Networks (ANNs) for function approximation. This approach effectively managed the game's large state space, enabling agents to progressively improve their gameplay. However, the study's focus on a specific card game might limit the broader applicability of its findings.

Next, we explore another variation of MC, **MC with importance sampling**, an off-policy learning algorithm.

2.3.6 MC with Importance Sampling

MC importance sampling is a technique designed to enhance the efficiency of MC simulations when estimating expected values. Instead of sampling directly from the original distribution, it draws samples from a proposal distribution. As illustrated in Alg. 5, the samples are re-weighted based on the ratio of the original distribution to the proposal distribution. This re-weighting process corrects the bias introduced by sampling from an alternative distribution, enabling more accurate and efficient estimations [42].

Next, we examine several studies that have applied MC importance sampling.

In [43], the authors proposed a non-iterative method for estimating the parameters of superimposed chirp signals in noisy environments using MC importance sampling. The approach employed maximum likelihood estimation to optimize the process efficiently, offering an alternative to traditional grid-search methods. The method focused on estimating chirp rates and frequencies, providing a computationally efficient solution for multidimensional problems. By eliminating iterative steps, it reduced computational complexity while maintaining high accuracy, even in challenging conditions. Additionally, the technique was scalable to multiple signal scenarios without significantly increasing computational demands. However, performance could degrade below certain signal-to-noise ratio thresholds, and problem-specific tuning of MC samples and parameters was necessary.

In [44], a simulation-based method was introduced for estimating blocking probabilities in multicast loss systems, enhancing static MC methods with importance sampling. The

Algorithm 5 MC with Importance Sampling [3]

Initialize, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
  $Q(s,a) \leftarrow$ arbitrary
  $C(s,a) \leftarrow 0$
  $\mu(a|s) \leftarrow$ an arbitrary soft behavior policy
  $\pi(a|s) \leftarrow$ an arbitrary target policy
repeat ▷ For ever
  Generate an episode using μ: $(S_0, A_0, R_1, \ldots, S_T, A_T, R_T, S_T)$
  $G \leftarrow 0$
  $W \leftarrow 1$
  for $t = T-1, T-2, \ldots, 0$ down to 0 **do**
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}\big[G - Q(S_t, A_t)\big]$
    $W \leftarrow W \cdot \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}$
    if $W = 0$ **then**
      break ▷ Exit inner loop
    end if
  end for
until convergence or a stopping criterion is met

approach decomposed the complex problem into simpler subproblems by isolating the contributions of individual network links to the blocking probability. A tailored distribution was used for each link's blocking states, and an inverse convolution method for sample generation, combined with a dynamic control algorithm, significantly reduced variance. While the method effectively allocated samples and improved variance reduction, its complexity and setup requirements could present barriers compared to more straightforward approaches.

The study by [45] developed an important sampling technique to improve simulation efficiency for rare-event phenomena. Initially designed for Markovian random walks, the method was later extended to accommodate non-Markovian processes and molecular dynamics simulations. By optimizing the sampling of successful transition paths, the approach enhanced computational efficiency. Additionally, a strategy for identifying an optimal importance function was introduced. However, optimizing this function remained challenging, especially in systems with extensive state spaces or those deviating from Markovian assumptions.

In [46], the authors presented an advanced MC importance sampling approach for evaluating the reliability of stochastic flow networks. The method utilized a recursive state-space decomposition to efficiently target critical network segments, reducing computational costs and improving reliability estimate precision. However, its applicability was potentially limited in very large or highly complex networks due to its dependence on specific network attributes.

In the next subsection, we introduce TD learning, another major category of tabular model-free methods and one of the fundamental concepts in RL.

2.3.7 Temporal Difference (TD) Learning

TD methods are regarded as a hybrid of MC methods and DP. Like MC approaches, TD learning enables agents to learn directly from raw experience without requiring a model of the environment's dynamics. At the same time, TD algorithms share similarities with DP, as they refine predictions using previously learned estimates rather than waiting for a final outcome to update values (a process known as bootstrapping). The following discussion introduces and analyzes various studies on TD-based approaches to provide a broad understanding of these algorithms before examining their specifics. The discussion begins with TD(0) and progresses to N-step SARSA, which will be covered in detail later in this survey.

2.3.8 TD(0)

Alg. 6 describes tabular TD(0) for estimating V_π. While MC methods require waiting until the conclusion of an episode to calculate the update to $V(S_t)$ as G_t, TD methods only need to wait until the subsequent timestep. At time $t+1$, TD methods swiftly establish a target and perform an effective update using the observed reward R_{t+1} and the estimated value $V(S_{t+1})$. It is important to recognize that the expression within the brackets in the TD(0) update represents a type of error. This error measures the discrepancy between the estimated value of S_t and a more refined estimate, $R_{t+1} + \gamma V(S_{t+1})$. This discrepancy is known as the TD error. The most basic form of the TD method executes this update as follows (line 7 in Alg. 6):

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \tag{2.12}$$

Upon transitioning to S_{t+1} and receiving R_{t+1}, TD methods implement the update immediately. The target for an MC update is G_t, while the target for the TD update is

Algorithm 6 Tabular TD(0) [3]

```
Initialize the value function V(s) arbitrarily (e.g., V(s) = 0 for all s ∈ S⁺)
repeat                                                    ▷ For each episode
    Initialize state S
    repeat                                                ▷ For each step of the episode
        A ← action given by policy π for S
        Take action A; observe reward R and next state S′
        V(S) ← V(S) + α[R + γV(S′) − V(S)]
        S ← S′
    until S is terminal
until convergence or for a specified number of episodes
```

$R_{t+1} + \gamma V(S_{t+1})$. This TD method is called TD(0), or one-step TD. TD(0) is a bootstrapping method, like DP, as its update is in part on an existing estimate [3].

TD(0) serves as a foundational concept for understanding other TD learning algorithms, including Q-learning, SARSA, and double Q-learning, which are examined in detail in the following subsections.

2.3.9 TD(0)-Replay

TD(0)-replay algorithm (Alg. 7), introduced in [47], enhances policy learning efficiency by leveraging full experience replay. Utilizing the agent's entire history of interactions allows for comprehensive updates to the value function and policy at each step. δ_t refers to the the TD error, R_{t+1} is the reward signal, γ is a discount factor, ϕ_t^T is the transpose of feature vector ϕ_t obtained through current state S_t, θ_t^T is the transpose of the weight vector θ_t, and α_t is a learning step; all varies according to time step t.

This approach accelerates convergence to optimal policies, particularly in environments where acquiring new experiences is costly or challenging. By efficiently replaying past experiences, **TD(0)-replay** proves invaluable in domains where re-experiencing events is infeasible or expensive. The method facilitates rapid optimization of the agent's value function, leading to faster learning and adaptation in dynamic environments, thereby offering significant advantages over approaches that lack replay mechanisms. The algorithm's adaptability is further demonstrated through dynamic updates of weight parameters based on historical data, ensuring effectiveness as environmental conditions evolve [47]. Its broad applicability across various domains highlights its versatility, making it suitable for both theoretical research and practical implementations.

Algorithm 7 TD(0)-Replay [47]

Input: $\alpha, \gamma, \theta_{\text{init}}$
$\theta \leftarrow \theta_{\text{init}}$
loop ▷ Over episodes
  Obtain initial state S, features ϕ
  $\hat{e} \leftarrow I_{n\times n}$, $e \leftarrow \mathbf{0}_{n\times 1}$
  while (terminal state has not been reached) **do**
    Act according to the policy
    Observe next reward $R = R_{t+1}$, next state $\hat{S} = S_{t+1}$, and its features $\hat{\phi} = \phi_{t+1}$
    $\alpha \leftarrow \ell(\alpha)$
    $e \leftarrow e + \alpha\phi\left(\left(\gamma\hat{\phi} - \phi\right)^\top e + R\right)$
    $\hat{e} \leftarrow \hat{e} + \alpha\phi\left[\left(\gamma\hat{\phi} - \phi\right)^\top \hat{e}\right]$
    $\theta \leftarrow \theta + \hat{e} + e$
    $\phi \leftarrow \hat{\phi}$
  end while
end loop

2.3.10 TD(λ)

TD(λ) is a powerful and flexible algorithm introduced in [48], extending the simpler TD(0) method by incorporating a parameter λ that controls the weighting of n-step returns. By blending one-step updates with MC methods, TD(λ) enables the algorithm to account for entire trajectories of experiences rather than only the immediate next step. A key feature of TD(λ) is the eligibility trace, which maintains a record of visited states and their relevance to learning updates. This mechanism addresses the **credit assignment problem**, helping determine how many past or future steps should contribute to an update. Algorithm 8 provides a comprehensive representation of the TD(λ) algorithm. The eligibility trace (lines 9-11) acts as a temporary record of events, such as state visits, and decays over time, governed by the parameter λ. This decay mechanism enables the algorithm to assign credit to prior states in a temporally distributed manner, effectively capturing delayed reward dependencies.

The value update in TD(λ) is given by (line 13):

$$V(s) \leftarrow V(s) + \alpha\delta e(s) \tag{2.13}$$

In TD(λ), α represents the learning rate, δ denotes the TD error defined as $\delta = R +$

Algorithm 8 TD(λ) [3]

```
Initialize V(s) arbitrarily (but set to 0 if s is terminal)
repeat                                                    ▷ For each episode
    Initialize E(s) = 0, for all s ∈ S
    Initialize S
    repeat                                                ▷ For each step of episode
        A ← action given by π for S
        Take action A, observe reward R, and next state S'
        δ ← R + γV(S') − V(S)
        E(S) ← E(S) + 1                                   ▷ Accumulating traces
        or E(S) ← (1 − α)E(S) + 1                         ▷ Dutch traces
        or E(S) ← 1                                       ▷ Replacing traces
        for all s ∈ S do
            V(s) ← V(s) + αδE(s)
            E(s) ← γλE(s)
        end for
        S ← S'
    until S is terminal
until convergence or a stopping criterion is met
```

$\gamma V(s') - V(s)$, and $e(s)$ is the eligibility trace associated with state s. TD(λ) establishes a continuum between TD(0) (pure TD updates) and MC (full return) methods, often achieving greater efficiency by balancing the bias-variance trade-off. By enabling the rapid propagation of value estimates across the state space, TD(λ) enhances learning efficiency and accelerates convergence [48, 3].

Next, we examine a selection of studies that have applied TD(λ) variants to explore the practical applications of this algorithm.

In [49], the authors introduced KnightCap, a chess program that learned its evaluation function using TDLeaf(λ), a variation of TD(λ) integrated with game-tree search. By playing on the Free Internet Chess Server, KnightCap improved its rating from 1650 to 2150 within 308 games over three days. This integration of TDLeaf(λ) with game-tree search marked a significant advancement in combining RL with traditional AI techniques in chess. The approach effectively leveraged deep evaluation through game-tree search while utilizing TD learning to refine its evaluation function dynamically. KnightCap's rapid improvement demonstrated the benefits of online learning against diverse opponents, emphasizing the role of varied strategic encounters. The adaptation of TD(λ) for minimax search allowed iterative improvement based on real-game outcomes, showcasing the algorithm's potential

in interactive environments. However, while the results were promising, further validation across varied game scenarios and against stronger AI opponents would provide a more comprehensive evaluation.

In [50], the authors extended the convergence theorem for TD learning to handle arbitrary time steps, which was critical for predicting future outcomes from past states. This theoretical advancement strengthened the foundation of TD(λ), ensuring convergence even in the presence of linearly dependent state representations. The work bridged the gap between TD learning and DP, reinforcing the strong convergence properties of TD methods. While primarily theoretical, the study provided strong guarantees regarding the stability and reliability of TD learning. Additional empirical validation in dynamic environments would further substantiate the proposed methods.

Building on prior theoretical work, [51] established a rigorous proof that TD(λ) algorithms converge with probability 1 under specific conditions. Expanding upon the earlier contributions of Sutton and Watkins, the authors formulated a general proof, ensuring reliable and consistent predictions. This theoretical advancement was instrumental in fostering the broader adoption of TD learning, as it provided strong assurances regarding stability and reliability. Although the paper remained theoretical, its detailed analysis significantly contributed to a deeper understanding of TD learning dynamics, informing future research directions.

In [52], the authors introduced QV(λ)-learning and the Actor-Critic Learning Automaton (ACLA), both extensions of TD(λ). QV(λ)-learning combined value function estimation with aspects of Q-learning, while ACLA applied a learning automaton-inspired update rule to the actor component. These algorithms were tested in various environments, demonstrating robustness and generalizability. QV(λ)-learning effectively merged the strengths of traditional Q-learning and actor-critic frameworks, resulting in improved stability and efficiency. ACLA introduced a flexible mechanism for adjusting action preferences, making it adaptable to diverse environments. Empirical results showed that these methods outperformed Q(λ)-learning, SARSA(λ), and traditional actor-critic approaches, particularly in learning speed and final performance. However, further validation in dynamic real-world applications would be necessary to assess their full potential in complex and partially observable domains.

In the next subsection, we examine another variant of the TD method: N-step bootstrapping.

2.3.11 N-step Bootstrapping

Having discussed TD(0)-replay and TD(λ), we now have a strong foundation for exploring bootstrapping methods. N-step bootstrapping extends the concept of value function updates by incorporating multiple steps rather than relying solely on immediate rewards or the value of the next state. This method strikes a balance between MC approaches, which defer updates until the end of an episode, and one-step TD methods, which update values immediately based on the next state. In N-step bootstrapping, the value function update is determined by *n subsequent rewards* and the estimated value of the state following these n steps. A key advantage of this approach is its potential to accelerate learning while reducing variance in updates compared to one-step methods [53, 3, 48]. The general update rule for the N-step method is given as:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[\sum_{k=0}^{n-1} \gamma^k R_{t+k+1} + \gamma^n V(S_{t+n}) - V(S_t) \right] \tag{2.14}$$

where R_{t+k+1} are the rewards received after taking action A_t from state S_t, γ is the discount factor, and α is the learning rate.

2.3.12 N-step TD Prediction

As previously discussed, N-step bootstrapping serves as the foundation for various adaptations, including N-step TD and N-step SARSA. N-step methods ($n \neq 1$) differ by incorporating multiple future steps, allowing for flexibility based on specific requirements. This approach contrasts with MC methods, which update state values based on the full sequence of observed rewards until the episode concludes, and one-step TD methods, which rely solely on the next reward and use the value of the state one step ahead as an estimate for future returns [54]. This distinction can be bridged by updating values based on an intermediate number of rewards—more than one, but fewer than the total number of future rewards in the episode. For instance, a two-step update incorporates the first two rewards along with the estimated value of the state two steps ahead. This concept extends naturally to three-step, four-step, and higher-order updates, progressively incorporating additional rewards and future state values.

Alg. 9 gives a general overview of the N-step TD prediction algorithm. For a given state S_t at time t, the update rule for the value function V in N-step TD prediction is (line 21):

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(n)} - V(S_t)) \tag{2.15}$$

Algorithm 9 N-step TD for Estimating $V \approx v_\pi$ [3]

1: Input: a policy π
2: Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
3: Initialize $V(s)$ arbitrarily for all $s \in S$
4: All store and access operations (for S_t and R_t) use index mod $n + 1$
5: **repeat** ▷ Loop for each episode
6: Initialize and store S_0 such that $S_0 \neq$ terminal
7: $T \leftarrow \infty$
8: **for** $t = 0, 1, 2, \ldots$ **do**
9: **if** $t < T$ **then**
10: Take an action according to $\pi(\cdot|S_t)$
11: Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
12: **if** S_{t+1} is terminal **then**
13: $T \leftarrow t + 1$
14: **end if**
15: **end if**
16: $\tau \leftarrow t - n + 1$ ▷ τ is the time whose state's estimate is being updated
17: **if** $\tau \geq 0$ **then**
18: $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
19: **if** $\tau + n < T$ **then**
20: $G \leftarrow G + \gamma^n V(S_{\tau+n})$
21: **end if**
22: $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
23: **end if**
24: **end for**
25: **until** $\tau = T - 1$

where α is the learning rate, and $G_t^{(n)}$ is derived as follows (line 18):

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) \tag{2.16}$$

Next, we review research studies that have implemented these methods, evaluating their advantages and limitations.

In [55], the authors introduced an advanced generalization of TD learning by incorporating networks of interrelated predictions. This method extended traditional TD techniques by linking multiple predictions over time, enabling a more structured approach to learning and representation. TD networks facilitated the learning of interconnected predictions, significantly expanding the range of representable and learnable functions. This

advancement proved especially beneficial in solving problems that standard TD methods struggled with, as demonstrated through experiments on the random-walk problem and predictive state representations. Experimental results indicated that TD networks learned complex prediction tasks more efficiently than MC methods, particularly in terms of data efficiency and learning speed. The practical advantages of TD networks were evident in environments requiring specific sequences of actions. While the study focused on small-scale experiments, the results suggested that TD networks had considerable potential for broader applications.

In [53], the authors introduced an advanced multi-step TD learning technique that utilized per-decision control variates to reduce variance in updates. This approach significantly improved the performance of multi-step TD algorithms, particularly in off-policy learning settings, by enhancing stability and convergence speed. Empirical results from tasks such as 5x5 Grid World and Mountain Car demonstrated that N-step SARSA outperformed N-step Expected SARSA by reducing root-mean-square error and improving learning efficiency. The method was also compatible with function approximation, reinforcing its practicality in complex environments.

Researchers in [56] developed a unified framework to analyze the finite-sample convergence guarantees of various value-based asynchronous RL algorithms. By reformulating these algorithms as Markovian stochastic approximation methods and applying Lyapunov analysis, the authors derived mean-square error bounds for their convergence. This framework systematically addressed convergence properties in algorithms such as Q-learning, N-step TD, TD(λ), and off-policy TD algorithms like V-trace. The study effectively tackled challenges related to asynchronous updates and provided robust convergence guarantees using detailed finite-sample mean-square bounds.

In [57], the authors investigated the deadly triad in RL—off-policy learning, bootstrapping, and function approximation—through a rigorous theoretical study of multi-step TD learning. Their work established a solid theoretical foundation for understanding these algorithms in off-policy settings with linear function approximation. A key contribution was the introduction of model-based deterministic counterparts to multi-step TD learning algorithms, improving the robustness of the analysis. The findings demonstrated that multi-step TD learning algorithms could converge to meaningful solutions when the sampling horizon n was sufficiently large, mitigating divergence issues observed in certain conditions.

In [58], the authors explored multiple facets of TD learning, particularly focusing on multi-step methods. The study introduced the use of control variates in multi-step TD learning to reduce variance in return estimates, thereby improving both learning speed and accuracy. The work also presented a unified framework for multi-step TD methods,

extending the N-step Q(σ) algorithm and proposing novel variations, including N-step CV Q(σ) and Q(σ, λ). This unification clarified the relationships between different multi-step TD algorithms and their variants. Additionally, the study extended predictive knowledge representation into the frequency domain, allowing TD learning agents to detect periodic structures in return sequences, leading to a richer understanding of the environment.

In [59], the authors introduced Undelayed N-step TD prediction (TD-P), which integrated eligibility traces, value function approximators, and environmental models. This method employed Neural Networks (NNs) to predict future steps within a learning episode, combining RL techniques to enhance learning efficiency and performance. By incorporating a forward-looking mechanism, the TD-P method aimed to gather additional information that backward-looking eligibility traces might overlook, leading to more precise value function updates and improved decision-making. The TD-P method was particularly well-suited for partially observable environments, where it leveraged NNs to handle continuous and complex state-action spaces effectively.

To further deepen our understanding of TD methods, we now analyze the off-policy variant of N-step learning in the following discussion.

2.3.13 N-step Off-policy Learning

N-step off-policy learning is an advanced technique that integrates N-step returns with off-policy updates. This method enhances learning stability and performance by utilizing multi-step returns while allowing data collection from a behavior policy b that differs from the target policy π, which is being improved [3]. Unlike one-step methods, multi-step returns aggregate rewards over multiple steps, providing a broader perspective on future rewards and yielding more precise value estimates. Importance sampling is employed to correct discrepancies between the behavior policy and the target policy. By adjusting updates with importance sampling ratios, the method accounts for differences in action probabilities between the two policies, ensuring accurate value estimation in off-policy settings.

The N-step off-policy return, $G_t^{(n)}$, can be calculated in the following way:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) \tag{2.17}$$

To maintain off-policy updates, importance sampling ratios are applied as follows:

$$G_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k \left(\prod_{i=0}^{k-1} \rho_{t+i} \right) R_{t+k+1} + \left(\prod_{i=0}^{n-1} \rho_{t+i} \right) \gamma^n V(S_{t+n}) \tag{2.18}$$

where $\rho_{t+i} = \frac{\pi(A_{t+i}|S_{t+i})}{b(A_{t+i}|S_{t+i})}$ is the importance sampling ratio, with π as the target policy and b the behavior policy [60].

With a solid foundation in place, we now conduct a detailed analysis of research papers to examine N-step off-policy methods. In [61], the authors introduced the Greedy Multi-step Value Iteration (GM-VI) algorithm, which approximates the optimal value function using a novel multi-step bootstrapping approach. The method dynamically adjusts the step size along each trajectory based on a greedy principle, effectively balancing information propagation and estimation accuracy. The adaptive step-size mechanism modifies step sizes according to the quality of trajectory data, improving learning efficiency and robustness. Unlike traditional off-policy methods, GM-VI learns safely from arbitrary behavior policies without requiring importance sampling corrections, simplifying the algorithm and reducing variance. Theoretical analysis demonstrated that GM-VI converged more rapidly to the optimal value function compared to one-step Bellman optimality operators. Empirical results on standard RL benchmarks, such as Mountain Car and Acrobot, confirmed that GM-VI achieved state-of-the-art sample efficiency and superior reward performance.

The core contribution of [62] was its elimination of importance sampling ratios in multi-step TD learning. By removing these ratios, the method reduced estimation variance, leading to improved stability and efficiency in off-policy learning. The introduction of action-dependent bootstrapping parameters further enhanced adaptability across different state-action pairs, refining update precision and reducing variance. Empirical evaluations on complex off-policy tasks demonstrated the algorithm's stability and superior performance relative to state-of-the-art methods, highlighting its practical applicability in real-world learning environments.

In [63], the authors proposed bias management techniques for multi-step Goal-Conditioned RL (GCRL), categorizing and mitigating shooting and shifting biases. This strategy enabled the use of larger step sizes, improving learning efficiency and overall performance. The proposed approach was validated across various tasks, consistently outperforming baseline multi-step GCRL benchmarks. The integration of quantile regression for bias handling further demonstrated the practical effectiveness of the approach. Additionally, the introduction of resilient bias-management strategies ensured robust learning improvements, with empirical results confirming its success across diverse environments.

In the following discussion, we provide a detailed analysis of Q-learning, one of the most widely utilized algorithms in RL.

Algorithm 10 Q-learning [3]

1: Initialize $Q(s, a)$, $\forall s \in S, a \in A(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$
2: **repeat** ▷ For each episode
3: Initialize S
4: **repeat** ▷ For each step of episode
5: Choose A from S using policy derived from Q (e.g., ϵ-greedy)
6: Take action A, observe R, S'
7: $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
8: $S \leftarrow S'$
9: **until** S is terminal
10: **until** convergence or a stopping criterion is met

2.3.14 Q-learning

Building upon TD(0), a significant breakthrough in RL was introduced by [9] with Q-learning, a model-free algorithm classified as off-policy TD control. Q-learning allows an agent to estimate the value of an action in a given state through direct experience without requiring a model of the environment. It is based on learning an action-value function, which represents the expected utility of selecting a particular action in each state and subsequently following a fixed policy. A general overview of the Q-learning algorithm is presented in Alg. 10. At its core, Q-learning updates Q-values (state-action value pairs), where the learned function Q approximates q_*, the optimal action-value function, independently of the policy being followed. This independence simplifies the algorithm's analysis and has facilitated early convergence proofs. However, the policy still influences the learning process by determining which state-action pairs are visited and subsequently updated (lines 4-9).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\big] \tag{2.19}$$

With a foundational understanding of Q-learning established, we now turn our attention to examining research papers that have applied this algorithm in various contexts. Q-learning, being fundamental and relatively straightforward, has been extensively applied across numerous studies. Here, we briefly touch upon a variety of notable studies.

Researchers in [64] investigated the performance differences between deterministic and stochastic policies within a grid-world problem using Q-learning. The authors developed a flexible agent capable of operating under both policy types to determine which parameters maximized cumulative reward. Their results indicated the superiority of deterministic

policies in achieving higher rewards in a structured task environment. The study's strength lies in its clear methodological execution, systematically exploring the impact of policy variations on learning outcomes. However, the study was confined to a simulated grid world, which may not fully capture the complexities of real-world environments, potentially reducing the generalizability of the findings. Additionally, the paper focused on policy optimization without significant consideration of the computational costs associated with each policy type.

In [65], the authors explored the development of a hardware architecture optimized for Q-learning, focusing on real-time applications. Key innovations included low power usage, high throughput, and minimal use of hardware resources. The implementation, tested on an evaluation kit, demonstrated improved performance metrics such as speed, power consumption, and hardware resource use, compared to existing Q-learning hardware accelerators. While the study presented a comprehensive approach to optimizing Q-learning for hardware implementation, making it suitable for real-time and Internet of Things (IoT) applications, it was limited by the specific hardware used for testing and the types of environments evaluated. The generalizability of the findings to other hardware or more complex real-world applications was not fully explored.

The application of Q-learning to improve power allocation in Wireless Body Area Networks (WBANs) was explored in [66]. The focus was on enhancing energy efficiency while maintaining effective communication within the network, particularly through controlling transmission power, reducing interference, and optimizing routing paths. The paper addressed critical aspects of WBANs that impact their practical deployment, especially in healthcare settings. While it presented a robust approach to managing interference and power consumption, the paper did not discuss the computational overhead introduced by the Q-learning algorithm and game theory applications, which was crucial for feasibility in devices with limited computational capabilities.

In [67], the authors explored improvements to the Q-learning algorithm's efficiency using massively parallel computing techniques, such as multi-threading and GPU computing. The paper addressed the issue of slow convergence in traditional Q-learning, particularly in complex, real-world system control problems that demanded quick adaptation to dynamic environments. By leveraging GPUs and multi-threading, the authors significantly decreased convergence time, which was vital for real-time applications like robotics and industrial control. However, the paper did not thoroughly examine how parallel processing affected the Q-learning algorithm's integrity and reliability, particularly under different computational loads.

Researchers in [68] investigated the use of Q-learning to automate portfolio re-balancing. The research applied basic Q-learning agents to discern trading patterns across 15 Indian

financial assets using technical indicators. The paper innovatively harnessed Q-learning to improve portfolio re-balancing, a domain where traditional rule-based systems might not perform optimally. However, the simplicity of the Q-learning models used might not have fully captured the intricate dependencies and dynamics of highly fluctuating markets.

Authors in [69] extended traditional Q-learning to handle POMDPs by incorporating L_0 regularization to manage the complexity of the state representation derived from the agent's history. This innovative approach transformed non-Markov and partially observable environments into a history-based RL framework, allowing Q-learning to be applicable in more complex scenarios. While the method showed improved computational and memory efficiency, the reliance on L_0 regularization could have made the learning process highly dependent on the initial choice of features, potentially limiting adaptability in dynamic environments.

An innovative approach to enhancing the performance of Proportional Integral Derivative (PID) controllers for magnetic levitation train systems through the integration of Q-learning was introduced in [70]. This method adapted PID parameters in real time, maintaining optimal levitation control despite non-linearities and uncertainties. While it showed improved performance over traditional PID controllers, the requirement for continuous learning and adjustment could be computationally intensive.

A novel application of Deep Q-learning (discussed in subsection 3.2.1) to optimize parameter settings in Hadoop, improving its efficiency by iteratively adjusting configurations based on feedback from performance metrics introduced in [71]. This approach reduced the manual effort and expertise required for parameter tuning and demonstrated marked improvements in processing speeds. However, the model's simplification of the parameter space might have overlooked interactions that could achieve further optimization, and the scalability and adaptability in varying operational environments remained somewhat uncertain.

Researchers in [72] presented a novel approach to enhancing the accuracy of nuclei segmentation in pathological images using Q-learning and Deep Q-Network (DQN) (will be discussed in 3.2.1) algorithms. The study reported improvements in Intersection over Union for segmentation, critical in cancer diagnosis. While the approach adapted the segmentation threshold dynamically, leading to better outcomes, the reliance on RL introduced computational complexity and required significant training data.

The sampling efficiency of Q-learning was discussed in [73], exploring the feasibility of making model-free algorithms as sample-efficient as model-based counterparts. The research introduced a variant of Q-learning that incorporated UCB exploration, achieving a regret bound that approached the best possible by model-based methods. This paper

marked a significant theoretical advance by providing rigorous proof that Q-learning could achieve sublinear regret in episodic MDPs without requiring a simulator. However, the analysis relied on assumptions such as an episodic structure and precise model knowledge, which might not have translated well to more uncertain environments.

Researchers in [74] examined the deployment of Q-learning on Field Programmable Gate Arrays (FPGAs) to enhance processing speed by leveraging parallel computing capabilities. The study demonstrated substantial acceleration of the Q-learning process, making it suitable for real-time scenarios. However, the focus on a particular FPGA model might have restricted the broader applicability of the findings to different hardware platforms.

The challenge of managing frequent handovers in high-speed railway systems using a Q-learning-based approach was addressed in [75]. The authors proposed a scheme that minimized unnecessary handovers and enhanced network performance by dynamically adjusting to changes in the network environment. While the approach showed promise, its complexity, computational demands, and need for real-time processing presented challenges for practical implementation.

In [76] authors explored optimizing taxi dispatching and routing using Q-learning and MDPs to enhance taxi service efficiency in urban environments. The method enabled dynamic adjustments based on real-time conditions, potentially alleviating urban traffic congestion. However, the model's effectiveness heavily depended on the accuracy and comprehensiveness of input data, and the computational complexity might have posed challenges for real-time deployment.

The study [77] addressed the optimization of network routing within optical transport networks using a Q-learning-based algorithm under the software-defined networking framework. The approach improved network capacity and efficiency by managing routes based on learned network conditions. While the method outperformed traditional routing strategies, its scalability and computational complexity in real-world scenarios remained areas for further exploration.

In [78], authors proposed a Q-learning algorithm motivated by internal stimuli, specifically visual novelty, to enhance autonomous learning in robots. The method blended Q-learning with a cognitive developmental approach, making the robot's learning process more dynamic and responsive to new stimuli. While the approach reduced computational costs and enhanced adaptability, its scalability in complex environments was not thoroughly explored.

The use of Q-learning and double Q-learning (discussed in the next subsection) to manage the duty cycles of IoT devices in environmental monitoring was analyzed in [79]. The study demonstrated significant advancements in optimizing IoT device operations,

enhancing energy efficiency and operational effectiveness. However, the performance heavily depended on environmental parameters, and the increased memory requirements for double Q-learning might have posed challenges for memory-constrained IoT devices.

The study [80] presented a robust implementation of RL to address resource allocation challenges in Fog Radio Access Networks (Fog RAN) for IoTs. The use of Q-learning effectively adapted to changes in network conditions, improving network performance and reducing latency. While the approach showed promise, the reliance on Q-learning might not have fully captured the complexities of larger-scale IoT networks, and the exploration-exploitation trade-off could have led to suboptimal performance in highly dynamic environments.

Q-learning exhibits an overestimation bias due to the max operator in its Q-value update rule, which inherently favors overestimated action values. To mitigate this issue, researchers developed a modified architecture, which is examined in the next subsection.

2.3.15 Double Q-learning

Double Q-learning, introduced by [81], enhances traditional Q-learning by mitigating overestimation bias, which can be problematic in environments with stochastic rewards. The overestimation arises because Q-learning uses the maximum action value as an approximation for the maximum expected action value [82]. To address this, as illustrated in Alg. 11, double Q-learning maintains two separate estimators (Q-tables), denoted as Q_A and Q_B (line 1). Each estimator is updated independently using the maximum value from the other estimator, thereby reducing the overestimation bias that standard Q-learning exhibits (lines 3-10).

The core update rule for double Q-learning involves: Selecting an action a based on the average of Q_A and Q_B. Updating one of the Q-functions at random with equal probability (0.5), for instance, updating Q_A using (line 7):

$$Q_A(s,a) \leftarrow Q_A(s,a) + \alpha\Big(r + \gamma Q_B\left(s', \arg\max_a Q_A(s',a)\right) - Q_A(s,a)\Big) \tag{2.20}$$

where s' is the next state in the environment. Alternatively, we shall update Q_B similarly using (line 10):

$$Q_B(s,a) \leftarrow Q_B(s,a) + \alpha\Big(r + \gamma Q_A\left(s', \arg\max_a Q_B(s',a)\right) - Q_B(s,a)\Big) \tag{2.21}$$

Algorithm 11 Double Q-learning [81]

Initialize Q^A, Q^B, s
repeat
    Choose a based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
    Choose (e.g., random) either UPDATE(A) or UPDATE(B)
    if UPDATE(A) **then**
        Define $a^* = \arg\max_a Q^A(s',a)$
        $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\big(r + \gamma Q^B(s',a^*) - Q^A(s,a)\big)$
    else if UPDATE(B) **then**
        Define $b^* = \arg\max_a Q^B(s',a)$
        $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)\big(r + \gamma Q^A(s',b^*) - Q^B(s,a)\big)$
    end if
    $s \leftarrow s'$
until end

Essentially, this approach alternates updates between Q_A and Q_B, using the maximum action value from the other table. This strategy is considered to provide a less biased estimate of the underlying value function [82]. By occasionally underestimating the maximum expected values, this method mitigates the positive bias inherent in Q-learning, aiming for a balance that better approximates true expectations.

In the following paragraphs, we explore a selection of research papers that have applied double Q-learning.

The focus of [83] was on the FPGA-based implementation of the double Q-learning algorithm, emphasizing its efficiency in stochastic environments and its superiority over standard Q-learning due to reduced overestimation biases. This implementation of FPGA allowed for the parallel processing of actions, significantly speeding up the learning process. This was particularly beneficial for applications requiring real-time decision-making. Additionally, the use of asynchronous reading and synchronous writing memory architectures optimized data exchange and reduced the hardware footprint, which was a critical advancement in hardware implementations of RL algorithms. However, while the hardware implementation was efficient for the stated range of states (from 8 to 256 states), the paper did not extensively discuss scalability beyond this range, which might be a limitation for environments with a much larger state space. The paper focused on the hardware aspect without a detailed discussion of how the implementation could be adapted to different RL scenarios or environments, which may have limited its applicability without additional modifications or tuning.

The integration of A* pathfinding with double Q-learning to optimize route planning in autonomous driving systems was investigated in [84]. This innovative approach aimed to enhance route efficiency and safety by minimizing the common problem of action value overestimations found in standard Q-learning. The paper introduced a novel combination of A* pathfinding with RL, providing a dual strategy that leveraged the deterministic benefits of A* for initial path planning and the adaptive strengths of the proposed method for real-time adjustments to dynamic conditions, such as unexpected obstacles. This synergy allowed for a balanced approach to navigating real-world driving scenarios efficiently. By employing double Q-learning, the system addressed and mitigated the issue of action value overestimation, which was prevalent in RL. This enhancement was crucial for autonomous driving applications where decisions had to be both accurate and dependable to ensure safety and operational reliability. On the other hand, this combination, while robust, introduced a significant computational demand that might have impacted the system's performance in real-time scenarios. Quick decision-making was essential in dynamic driving environments, and the increased computational load could have hindered the system's ability to respond promptly. Additionally, the promising method introduced lacked a detailed exploration of how this hybrid model performed across different environmental conditions or traffic scenarios. This limitation might have affected the model's effectiveness in diverse settings without further adaptation or refinement.

Researchers in [85] delved into optimizing IoT devices' power management by employing a double Q-learning-based controller. This Double Data-Driven Self-Learning (DDDSL) controller dynamically adjusted operational duty cycles, leveraging predictive data analytics to enhance power efficiency significantly. A notable strength of the paper was the improved operational efficiency introduced by the double Q-learning, which effectively handled the overestimation issues found in standard Q-learning within stochastic environments. This led to more precise power management decisions, crucial for prolonging battery life and minimizing energy usage in IoT devices. Furthermore, the DDDSL controller showed a marked performance enhancement, outperforming traditional fixed duty cycle controllers by 42–50%, and the previous data-driven self-learning model by 2–12%. However, while the performance improvements were compelling, they were obtained under specific conditions, which might have limited the broader applicability of the findings without further adaptations or validations for different operational environments or IoT device configurations.

A double Q-learning-based routing protocol for optimizing maritime network routing, crucial for effective communication in maritime search and rescue operations, was developed in [86]. The use of double Q-learning in this context aimed to tackle the overestimation problems inherent in Q-learning protocols, which was a significant improvement

in maintaining stability in the model's predictions and actions. This approach not only enhanced the routing efficiency but also incorporated a trust management system to ensure the reliability of data transfers and safeguard against packet-dropping attacks. The protocol demonstrated robust performance in various simulated attack scenarios with efficient energy consumption and minimal resource footprint, crucial for the resource-constrained environments in which maritime operations occur. While the proposed method showed promising results in simulations, the complexity of real-world application scenarios could have posed challenges. Maritime environments were highly dynamic with numerous unpredictable elements, which might have affected the consistency of the performance gains observed in controlled simulations. Additionally, the scalability of this approach, when applied to very large-scale networks or under extreme conditions typical of maritime emergencies, could have required further validation. The integration of such sophisticated systems also raised concerns about the computational overhead and the practical deployment in existing maritime communication infrastructures.

Authors in [87] explored the application of double Q-learning to manage adaptive wavelet compression in environmental sensor networks. This method aimed to optimize data transmission efficiency by dynamically adjusting compression levels based on real-time communication bandwidth availability. A key advantage of this approach was its high adaptability, which ensured efficient bandwidth utilization and minimized data loss even under fluctuating network conditions, critical for remote environmental monitoring stations where connectivity might have been inconsistent. Furthermore, this integration helped significantly reduce the risk of overestimating action values, a common problem in Q-learning, which could have led to suboptimal compression settings. However, the complexity of this implementation posed a notable challenge, particularly in environments where computational resources were limited. The necessity for managing two separate Q-values for each action increased the computational demand, potentially impacting the system's real-time response capabilities. Additionally, the performance of the algorithm heavily depended on the accuracy of the network condition assessments, with inaccuracies potentially leading to inefficient compression and data transmission.

In [88], authors explored the application of double Q-learning to enhance dynamic voltage and frequency scaling in multi-core real-time systems. Their approach addressed the inherent overestimation bias found in Q-learning, aiming to provide more accurate and reliable power management. A major strength of their work lies in the innovative use of double Q-learning to mitigate the overestimation issue common in single-estimator Q-learning methods. By utilizing two estimators, the system could potentially make more informed decisions that better balanced power consumption against system performance needs. Additionally, their thorough simulation-based evaluation indicated that this method

could outperform traditional methods, offering substantial energy savings across various system conditions. Nevertheless, managing two Q-value estimators increased the computational load, which could have challenged the limited resources available in real-time systems where rapid processing was paramount. Furthermore, while the simulation results were decent, translating these outcomes to real-world scenarios required additional adjustments to accommodate the diverse nature and unpredictability of real system workloads.

Having examined the off-policy TD algorithm Q-learning and its variant double Q-learning, which employs two Q functions, we now shift our focus to another TD-based algorithm, SARSA, in the next subsection.

2.3.16 State-Action-Reward-State-Action (SARSA)

The SARSA algorithm is an on-policy method that updates the action-value function (Q-function) incrementally and in real time. Introduced in [10], SARSA differs from other approaches by learning Q-values directly from the policy being executed. Unlike off-policy methods, such as Q-learning, which can learn the optimal policy regardless of the agent's actions, SARSA updates values based on the policy currently being followed. The core mechanism of SARSA lies in its Q-value update rule, which follows the equation defined in [3] and is implemented in Algorithm 12, line 8:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \tag{2.22}$$

Both SARSA and Q-learning update their estimates of the action-value function based on the agent's actions. However, the key distinction lies in how they update these values. Q-learning updates based on the maximum estimated future reward, using the highest Q-value of the next state, irrespective of the action taken. In contrast, SARSA updates its Q-values using the actual action taken in the next state rather than selecting the best possible action. In other words, SARSA does not wait to compute the maximum Q-value of the next action but directly incorporates the next available action into its update process.

Next, we examine several research studies that have applied the SARSA algorithm.

The application of the SARSA algorithm in a simulated shepherding scenario where a dog herds sheep towards a target was investigated in [89]. This RL model incorporated a discretized state and action space and designed a specific reward system to facilitate learning. The application of SARSA to a complex, dynamic task like shepherding demonstrated the algorithm's versatility and its potential in environments involving multiple agents and stochastic elements. The model successfully taught a dog to herd sheep by learning to

Algorithm 12 SARSA [3]

```
Initialize Q(s, a), ∀s ∈ S, a ∈ A(s), arbitrarily, and Q(terminal-state, ·) = 0
repeat                                                          ▷ For each episode
    Initialize S
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    repeat                                                      ▷ (for each step of the episode)
        Take action A, observe R, S′
        Choose A′ from S′ using policy derived from Q (e.g., ϵ-greedy)
        Q(S, A) ← Q(S, A) + α [R + γQ(S′, A′) − Q(S, A)]
        S ← S′; A ← A′
    until S is terminal
until convergence or a stopping criterion is met
```

reach sub-goals, which simplified the learning process and improved the manageability of the task. However, the discretization of the state and action spaces might have limited the dog's movement and decision-making capabilities, potentially leading to less optimal performance in more realistic or varied environments. The stochastic nature of sheep movement and the complexity of the shepherding task led to a significant learning time, with notable success only after many episodes, indicating potential efficiency issues in more demanding or time-sensitive applications.

Authors in [90] primarily focused on the implementation of Q-learning and SARSA algorithms, incorporating eligibility traces to improve handling delayed rewards. The research demonstrated the effectiveness of RL algorithms in navigating dynamic tasks like micro-managing combat units in real-time strategy games. Adding eligibility traces significantly boosted the algorithms' learning from sequences of interdependent actions, essential in fast-paced, chaotic environments. Using a commercial game like StarCraft as a testbed introduced real-world complexity often absent in simulated settings. This method confirmed the practicality of RL algorithms in real scenarios and highlighted their potential to adapt to commercial applications. The algorithms showed promise in small-scale combat, but scaling to larger, more complex battles in StarCraft or similar games remained uncertain. Concerns arose about computational demands and the efficiency of learning optimal strategies without extensive prior training. While effective within StarCraft, their broader applicability to other real-time strategy games or applications remained untested. The specialized design of state and action spaces for StarCraft could have hindered transferring these methods to different domains without significant modifications.

In [91], the authors studied the enhancement of energy efficiency in IoT networks

through strategic power and channel resource allocation using a SARSA-based algorithm. This approach addressed the unpredictable nature of energy from renewable sources and the variability of wireless channels in real-time settings. A major contribution of the study lies in its innovative use of a model-free, on-policy RL method to manage energy distribution across IoT nodes. This method efficiently handled the stochastic nature of energy harvesting and channel conditions, optimizing network performance in terms of energy efficiency and longevity. Integrating SARSA with linear function approximation helped refine solutions to continuous state and action spaces, enhancing the practicality in real-world IoT applications. However, relying on linear function approximation introduced limitations in capturing the full complexity of interactions in dynamic, multi-dimensional state spaces typical of IoT environments. While the proposed SARSA algorithm demonstrated network efficiency improvements, implementing such an RL system in real IoT networks posed challenges. These included the need for continual learning and adaptation to changing conditions, impacting the deployment's feasibility and scalability.

Authors in [92] explored applying SARSA to optimize peer-to-peer (P2P) electricity transactions among small-scale users in smart energy systems. This research aimed to enhance economic efficiency and reliability in decentralized energy markets. A significant strength lies in its innovative application of SARSA to a complex, dynamic energy trading system. Researchers modeled P2P electricity transactions as an MDP, allowing the system to handle uncertainties in small-scale energy trading, like fluctuating prices and varying demands. This modeling enabled the system to learn optimal transaction strategies over time, potentially enhancing both efficiency and profitability in energy trading. However, the approach had limitations. SARSA, while effective in learning optimal policies through trial and error, required extensive interaction with the environment to achieve satisfactory performance. This data requirement could have been a drawback in real-world applications where immediate decisions were necessary and historical transaction data was limited. Moreover, implementing such a system in a live environment, where real-time decision-making was crucial, posed additional challenges, including the need for robust computational resources to handle continuous state and action space calculations.

In [93], they examined integrating SARSA(0) with Fully Homomorphic Encryption (FHE) for cloud-based control systems to ensure data confidentiality while performing RL computations on encrypted data. A significant strength was preserving privacy in cloud environments where sensitive control data could have been vulnerable to breaches. Using FHE allowed the RL algorithm to execute without decrypting the data, providing a robust method for maintaining confidentiality. The paper successfully demonstrated this method on a classical pole-balancing problem, showing it was theoretically sound and practically feasible. However, implementing SARSA(0) over FHE introduced challenges related to

computational overhead and latency due to encryption operations. These factors could have impacted the efficiency and scalability of the RL system, especially in environments requiring real-time decision-making. Additionally, encryption-induced delays and managing encrypted computations might have limited this method's application to scenarios where control tasks could tolerate such delays.

Authors in [94] presented a novel application of SARSA within a swarm RL framework to solve optimization problems more efficiently, particularly those involving large negative rewards. The authors incorporated individual learning and cooperative information exchange among multiple agents, aiming to speed up learning and enhance decision-making efficiency. This approach significantly leveraged swarm intelligence and RL, particularly SARSA's ability to handle tasks with substantial negative rewards. By enabling agents to learn from individual experiences and shared insights, the system could converge to optimal policies more swiftly than traditional single-agent or non-cooperative multi-agent systems. Implementing the shortest path problem demonstrated the method's practicality and effectiveness, showcasing improved learning speeds and robustness against pitfalls with significant penalties. Nonetheless, managing multiple agents and their interactions increased the computational overhead and complexity of the learning process. Moreover, the approach heavily relied on designing the information-sharing protocol among agents, which, if not optimized, could have led to inefficiencies or suboptimal learning outcomes. The generalized application of this method across different environments or RL tasks remained to be thoroughly tested, suggesting potential limitations in adaptability.

In [95], authors explored the application of SARSA to optimize routing for Electric Vehicles (EVs) to minimize energy consumption. This approach adapted to real-time driving conditions to reduce on-road energy needs by selecting routes with lower energy requirements. The research's strength lies in its real-time application and use of SARSA to learn and predict the most energy-efficient routes under various conditions, like traffic and road type. By utilizing Markov chain models to estimate energy requirements based on actual driving data, the framework aimed to extend the driving range of EVs, which is crucial for reducing range anxiety. However, the reliance on real-time data and the inherent variability of driving conditions presented challenges. The accuracy of the SARSA model's predictions heavily depended on the data quality and immediacy, which could have been compromised by limitations in data transmission or processing delays. Implementing this system in a real-world environment could have posed scalability and adaptability challenges, particularly concerning integration with existing vehicle navigation systems and the computational demands on the vehicle's hardware.

A variation of SARSA that considers expectations over all possible next actions rather than relying solely on the sampled next action is introduced in the next subsection.

Algorithm 13 Expected SARSA [96]

```
Initialize Q(s, a) arbitrarily for all s, a
repeat                                                    ▷ over episodes
    Initialize s
    repeat                                                ▷ for each step in the episode
        choose a from s using policy π derived from Q
        take action a, observe r and s'
        V_{s'} = Σ_{a'} π(a'|s') · Q(s', a')
        Q(s, a) ← Q(s, a) + α [r + γ V_{s'} − Q(s, a)]
        s ← s'
    until s is terminal
until end of episodes
```

2.3.17 Expected SARSA

The Expected SARSA algorithm, introduced by [96], extends the classic SARSA algorithm by incorporating expectations over all possible next actions rather than relying solely on the sampled next action. This modification reduces the variance in the update rule, potentially leading to faster and more stable convergence in learning tasks. The key idea behind Expected SARSA is that averaging over all possible next actions, weighted by their probability under the current policy, provides a more stable estimate of state-action values. Expected SARSA ensures convergence under conditions similar to those required for SARSA, such as all state-action pairs being visited infinitely often. [96] formally proved that Expected SARSA converges to the optimal action-value function, assuming finite state and action spaces and a policy that becomes greedy in the limit with infinite exploration. Empirical results demonstrated that Expected SARSA outperformed both SARSA and Q-learning in various tasks, particularly those where certain actions could result in significant negative consequences. By integrating the expectation over all possible next actions, the algorithm smooths out learning updates, which is particularly advantageous in stochastic environments where certain actions or state transitions exhibit high variability in rewards [96, 3]. In summary, Expected SARSA provides a robust approach to RL by reducing the variance in state-action value updates, leading to more reliable learning performance, especially in complex and uncertain environments.

The Expected SARSA update rule is given as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \sum_a \pi(a|s_{t+1}) Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{2.23}$$

Before reviewing studies that have applied Expected SARSA, a detailed overview of the algorithm is provided in Alg. 13.

Authors in [97] designed a comprehensive exploration of combining Model Predictive Control (MPC) with the Expected SARSA algorithm to tune MPC models' parameters. This integration aimed to enhance the robustness and efficiency of control systems, particularly in applications where the system model's parameters were not fully known or subject to change. A key advantage was the innovative approach to speeding up learning by directly integrating RL with MPC, reducing the episodes needed for effective training. This efficiency was achieved using the Expected SARSA algorithm, which offered smoother convergence and better performance due to its average-based update rule compared to the more common Q-learning method, which focused on maximum expected rewards and might lead to higher variance in updates. However, the approach's complexity and the need for precise tuning of MPC model parameters represented significant challenges. The computational demands increased due to the dual needs of continuous adaptation by the RL algorithm and the rigorous constraints enforced by MPC. While the framework showed potential in simulations, its real-world applicability, especially in highly dynamic and unpredictable environments, required further validation.

Authors in [98] focused on improving Deterministic and Synchronous Multichannel Extension (DSME) networks' resilience against WiFi interference using the Expected SARSA algorithm. It evaluated channel adaptation and hopping strategies to mitigate interference in industrial environments, providing a detailed analysis with Expected SARSA. The study stood out for its rigorous simulation-based evaluation of interference mitigation strategies in a controlled DSME network. By using Expected SARSA, the research effectively reduced uncertainty in channel quality assessment, leading to more reliable network performance under interference conditions, crucial for industrial applications where reliable data transmission was vital for operational efficiency and safety. However, the complexity of RL implementation and its dependency on accurate real-time data posed challenges. The computational demands of running Expected SARSA in real-time environments could limit the practical deployment of this strategy in resource-constrained settings.

In [99], researchers investigated using the Expected SARSA learning algorithm to manage Load Frequency Control in multi-area power systems. The study focused on enhancing power systems' stability integrated with distributed feed-in generation-based wind power,

using a model-free RL approach to adjust to variable power supply conditions without a predefined system model. The primary strength lies in its innovative approach to addressing challenges in integrating renewable energy sources into the power grid. Expected SARSA, an on-policy RL algorithm, demonstrated how to manage and stabilize frequency variations due to unpredictable renewable energy outputs and fluctuating demand adaptively. This was crucial for maintaining reliability and efficiency in power grids increasingly incorporating variable renewable energy sources. However, the approach's downside was related to the RL algorithm's computational demands and the need for extensive simulation and testing to fine-tune system parameters. Implementing such a system in a real-world setting could be constrained by these factors, particularly in terms of real-time computation capabilities and the scalability of the solution to larger, more complex grid systems.

Authors in [100] utilized the Expected SARSA algorithm to optimize Energy Storage Systems (ESS) operation in managing uncertainties in wind power generation forecasts. The study modeled the problem as MDP where the state and action spaces were defined by the ESS's operational constraints. Expected SARSA's primary result in this context was its superior performance compared to conventional Q-learning-based methods. The algorithm effectively handled the wide variance in wind power forecasts, crucial for optimizing ESS's charging and discharging actions to reduce forecast errors. The strategy's effectiveness was underscored by its near-optimal performance, closely approximating the optimal solution with complete future information. Lastly, we examine the last variant of SARSA, namely N-step SARSA.

2.3.18 N-step SARSA

With the foundational concepts of n-step TD and bootstrapping established, we can expand on these ideas and explore research papers utilizing them. First, we explore N-step SARSA, an enhancement of the conventional one-step SARSA within the on-policy learning framework. In N-step SARSA, as shown in Alg. 14, the update to the value (or action-value) is not limited to just the next state and action, as seen in one-step SARSA, but instead incorporates a sequence of n actions and rewards. The value function in the equation is replaced with $Q(S_{t+n}, A_{t+n})$:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[\sum_{k=0}^{n-1} \gamma^k R_{t+k+1} + \gamma^n Q(S_{t+n}, A_{t+n}) - Q(S_t, A_t) \right] \tag{2.24}$$

Algorithm 14 N-step SARSA [3]

```
Initialize Q(s, a) arbitrarily, for all s ∈ S, a ∈ A
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Algorithm parameters: step size α ∈ (0, 1], small ε > 0, a positive integer n
All store and access operations (for S_t, A_t, R_t) can take their index mod n + 1
repeat                                                          ▷ For each episode
    Initialize and store S_0 ≠ terminal
    Select and store an action A_0 ~ π(·|S_0)
    T ← ∞
    for t = 0, 1, 2, ... do
        if t < T: then
            Take action A_t
            Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
            if S_{t+1} is terminal then
                T ← t + 1
            else
                Select and store an action A_{t+1} ~ π(·|S_{t+1})
            end if
        end if
        τ ← t − n + 1    (τ is the time whose estimate is being updated)
        if τ ≥ 0: then
            G ← Σ_{i=τ+1}^{min(τ+n,T)} γ^{i−τ−1} R_i
            if τ + n < T: then
                G ← G + γ^n Q(S_{τ+n}, A_{τ+n})    (G_{τ:τ+n})
            end if
            Q(S_τ, A_τ) ← Q(S_τ, A_τ) + α [G − Q(S_τ, A_τ)]
            if π is being learned then
                Ensure that π(·|S_τ) is ε-greedy with respect to Q
            end if
        end if
    end for
until τ = T − 1
```

N-step SARSA retains the on-policy nature of SARSA, allowing it to effectively combine the advantages of TD learning with the broader horizon of MC methods. By adjusting the

number of steps n ahead, it balances the bias-variance trade-off. N-step methods, including N-step SARSA, improve learning by incorporating multiple future outcomes rather than depending solely on the immediate next state and action. This leads to faster learning and enhances policy performance, particularly in complex environments where future rewards depend heavily on actions taken over multiple steps [3, 54].

In [101], the N-step SARSA algorithm was applied to optimize traffic signal control to dynamically adjust signals based on real-time conditions to reduce congestion more effectively than traditional methods like static signaling and longest queue First. The approach incorporated multiple future steps for more strategic planning, tested using the SUMO simulator with real-world traffic data from Texas. The study showed potential improvements in traffic flow management and scalability by using a centralized control agent to manage state-action space growth. However, challenges for real-world implementation include high computational demands, reliance on real-time data processing, and communication infrastructure. Additionally, the study's focus on a specific urban environment and the risk of overfitting to simulation conditions may affect the algorithm's generalizability across different traffic patterns or during events like accidents or road closures. Following, we examine the practical applications of this algorithm across different domains.

In [102], authors introduced a hybrid method that melded on-policy traits of SARSA with off-policy features of Q-learning, integrating an N-step look-ahead capability to enhance foresight in decision-making. The algorithm's flexibility to adjust the number of look-ahead steps (N) dynamically allowed it to suit different learning stages, providing an effective balance between exploration and exploitation. This adaptability could facilitate more efficient learning than traditional RL methods. By combining on-policy and off-policy updates, the algorithm capitalized on the strengths of both SARSA and Q-learning, potentially decreasing the overestimation bias seen in Q-learning while still targeting optimal policies. The approach's ability to adjust the N parameter based on learning phases or specific conditions suggested customization for a broad array of applications, from simple gaming scenarios to intricate decision-making tasks in real-world contexts. However, the algorithm's requirement to finely adjust various hyperparameters, like maximum and minimum values of N and the breakpoint for decrementing N, added complexity to its configuration and optimization. This could pose a challenge, particularly for users with limited RL experience. Additionally, maintaining and updating a combination of policies over multiple steps before reaching a decision resulted in higher computational demands, potentially restricting the algorithm's applicability in settings with limited resources or where rapid response times were essential.

2.4 Analytical Experiments

In this section, an analytical comparison between several tabular RL algorithms is given. The analyzed algorithms are Q-learning, Double Q-learning, SARSA, and Expected SARSA on a Maze Grid environment. The environment consists of a 10x10 grid of cells with various features that define the agent's interactions and objectives. The agent, represented as a red square, must navigate the grid to reach a goal, represented as a yellow oval. Reaching the goal yields a reward of 1, ending the current episode and resetting the environment. The grid includes pits (blue squares) that result in a penalty of -10 and terminate the episode if encountered. There are also walls (black squares), which are impassable; if the agent attempts to move into a wall, it remains in its current position and incurs a penalty of -0.3. Moving to any other cell besides the goal imposes a small penalty of -0.1, encouraging the agent to reach the goal state efficiently. Snapshots of the environment with different tasks are given in Fig. 2.2.

(a) Task 1 (b) Task 2 (c) Task 3

Figure 2.2: Maze Grid Environment

To manage exploration across all tasks, the epsilon-greedy approach was employed, specifically using a decaying epsilon strategy. Additionally, a decaying schedule was applied to the learning rate. The following table of hyperparameters outlines the parameters utilized for all methods:

- Learning rate (α): 0.1
- Discount factor (γ): 0.9

- Initial epsilon (ϵ): 0.2
- Minimum learning rate: 0.001
- Minimum epsilon: 0.01
- Decay rate: 0.99
- Number of episodes: 2000

The discount factor γ was set to 0.9 to prioritize achieving the goal state in the shortest time possible, aligning with the environment's objectives. The initial epsilon value was chosen to facilitate significant exploration, which is crucial in the early episodes. A decay rate of 0.99 ensured a gradual reduction in exploration, allowing the agent to focus more on exploitation in later episodes. This balance of exploration and exploitation was similarly maintained for the learning rate (α), with the minimum boundaries preventing ϵ and α from falling below their defined thresholds.

The results for Task 1 are presented in Fig. 2.3. All algorithms demonstrated improvements over time; however, SARSA and Expected SARSA exhibited more stable and consistent learning patterns compared to Q-Learning and Double Q-Learning. These SARSA-based methods showed faster convergence with fewer fluctuations, while Q-Learning-based methods were more erratic before stabilizing near zero. Double Q-Learning performed the worst among the methods.

In the path length analysis, all methods successfully reduced path lengths over time. Expected SARSA and SARSA, however, demonstrated superior performance with shorter and more consistent path lengths. While Double Q-Learning showed initial progress, its path lengths ultimately converged to levels similar to Q-Learning.

Figure 2.3: Task 1 Comparison

Experiment Setup:

- Episodes: 2000
- Simulation speed: 0.008

```
[Q-Learning on Task 1]
  max-rew = -0.900
  med-3 = -7.600
  var-3 = 9.016
  max-episode-len = 664.0

[Double Q-Learning on Task 1]
  max-rew = -0.300
  med-3 = -0.800
  var-3 = 0.016
  max-episode-len = 735.0
```

```
[SARSA on Task 1]
  max-rew = -0.700
  med-3 = -8.900
  var-3 = 4.809
  max-episode-len = 635.0

[Expected SARSA on Task 1]
  max-rew = -0.700
  med-3 = -5.700
  var-3 = 15.860
  max-episode-len = 806.0
```

In Task 2, as shown in Fig. 2.4, all algorithms improved over time, with SARSA-based methods again demonstrating superior stability and faster convergence. The differences in performance across all algorithms were less pronounced than in Task 1, but SARSA and Expected SARSA still outperformed the Q-Learning approaches. Double Q-Learning exhibited a higher variance, leading to suboptimal performance in reducing path lengths.

Figure 2.4: Task 2 Comparison

Experiment Setup:

- Episodes: 2000
- Simulation speed: 0.008

```
[Q-Learning on Task 2]
  max-rew = 0.300
  med-3 = 0.300
  var-3 = 0.000
  max-episode-len = 224.0

[Double Q-Learning on Task 2]
  max-rew = 0.200
  med-3 = -0.200
  var-3 = 0.327
  max-episode-len = 192.0

[SARSA on Task 2]
  max-rew = 0.300
  med-3 = 0.300
  var-3 = 0.020
  max-episode-len = 321.0

[Expected SARSA on Task 2]
  max-rew = 0.300
  med-3 = 0.300
  var-3 = 0.000
  max-episode-len = 349.0
```

In Task 3, Double Q-Learning exhibited substantial instability, especially in early episodes, as shown in Fig. 2.5. Sharp dips in rewards and high variance impeded its performance. By contrast, SARSA-based methods and Q-Learning displayed more consistent learning trends, with Expected SARSA emerging as the most stable and effective approach. Double Q-Learning's high variability contributed to its poor performance in both reward maximization and path length reduction.

Figure 2.5: Task 3 Comparison

Experiment Setup:

- Episodes: 2000
- Simulation speed: 0.008

```
[Q-Learning on Task 3]
  max-rew = -0.800
  med-3 = -9.100
  var-3 = 4.527
  max-episode-len = 400.0

[Double Q-Learning on Task 3]
  max-rew = -5.900
  med-3 = -155.400
  var-3 = 12362.327
  max-episode-len = 63586.0
```

```
[SARSA on Task 3]
  max-rew = -1.000
  med-3 = -7.800
  var-3 = 10.916
  max-episode-len = 422.0

[Expected SARSA on Task 3]
  max-rew = -0.200
  med-3 = -5.700
  var-3 = 32.327
  max-episode-len = 403.0
```

Overall, SARSA-based algorithms, particularly Expected SARSA, consistently outperformed Q-Learning-based approaches across all tasks, as expected and discussed earlier. Double Q-Learning exhibited significant instability and performed poorly in all scenarios. The results highlight the effectiveness of SARSA-based methods in environments requiring stable and efficient learning.

2.5 Conclusion

This chapter has explored the foundational principles of RL, providing a structured examination of its core methodologies. By delving into both model-based and model-free approaches, we highlighted the strengths and limitations of each method while addressing critical challenges in exploration, exploitation, and scalability. The detailed analyses of temporal-difference learning and Monte Carlo methods emphasize their theoretical significance and practical utility across various domains. As RL continues to evolve, integrating these foundational techniques with modern approaches, such as deep RL and Policy Optimization, is pivotal in addressing increasingly complex decision-making problems. This work provides a solid foundation for researchers and practitioners seeking to deepen their understanding of RL's core landscape.

Chapter 3

Deep Reinforcement Learning

In this chapter, we review DRL methods, ranging from approximation value-based methods to policy-based and actor-critic methods. We then analyze some of the algorithms in a highway traffic environment to test their capability and observe the behavior of agents trained with different algorithms.

3.1 Introduction

RL trains an agent to interact with its environment to achieve goals by maximizing cumulative rewards.The core framework involves components such as a policy, a reward signal, a value function, and, optionally, a model of the environment, with Markov Decision Processes (MDPs) providing the mathematical foundation for sequential decision-making. RL methods are generally divided into two categories: model-free and model-based approaches. Each offers unique advantages based on task complexity and the availability of environmental knowledge.A significant challenge in RL is balancing exploration (learning new strategies) and exploitation (using known strategies), with various strategies and techniques developed to manage this trade-off effectively [1, 103, 12]. For a detailed introduction to RL, readers are strongly referred to [104] since we provided a thorough and complete introduction to it, and our focus in this survey is on DRL. Over the next subsection, we give a brief introduction to deep learning to build a strong foundation to understand various algorithms.

3.1.1 Deep Learning Overview

Deep learning is a branch of ML within the broader scope of AI, where multiple-layer neural networks are utilized to model complex data patterns. with multiple layers are used for the modeling of complex data patterns. Drawing from the organization and operation of the human brain, deep learning employs artificial neural networks to analyze and learn from extensive data sets, facilitating sophisticated uses in image recognition, Natural Language Processing (NLP), and autonomous systems [105]. Deep learning can trace its roots to earlier studies on perceptrons [106] and multi-layer perceptrons that were foundational but, owing to computational constraints and problems of vanishing gradient in backpropagation, were limited [107]. The second wave of deep learning was further made possible by algorithmic advancements like the Rectified Linear Unit (ReLU) [108], dropout [109], large-scale datasets, and high-performance GPUs [110]. Deep learning architectures are distinguished by their complexity, which includes multiple layers. Notable examples are Convolutional Neural Networks (CNNs), which are widely utilized for image processing, and Recurrent Neural Networks, which encompass Long Short-Term Memory (LSTM) networks designed for handling sequential data. Transformer models ([111]) have further revolutionized the field, particularly in NLP, as seen in applications like BERT ([112]) and GPT ([113]). We refer you to [114, 105, 115, 116] to understand and learn deep learning in detail, as we expect readers to have at least basic knowledge in the mentioned architectures as they are being used in different DRL algorithms.

3.2 Approximation Value-based Methods

This section analyzes approximation value-based methods and their applications in various domains. Approximation model-free algorithms in RL comprise methods oriented to learning policies and value functions solely from interaction with an environment, though without an explicitly stated model of the environment's dynamics. These algorithms typically use function approximators, such as neural networks, and generalize from observed state–action pairs to unknown ones. Consequently, they are quite effective at handling large or continuous states and actions [117, 118, 119].

Key features of the approximation model-free algorithms are:

- These algorithms learn the policy directly by optimizing expected reward without an explicit construction of the model of the environment.
- In value-function estimation, the value function is estimated using function approximators predicting expected rewards for states or state–action pairs.

- The solutions are scalable, thus fitting for problems with large or continuous state and action spaces since they can generalize from limited data.

Popular examples include Q-learning algorithms with function approximation, called Deep Q-Networks (DQN). Approximation model-free algorithms are thus essential enablers of practical applications in RL where an explicit environment model is infeasible or too computationally expensive to be created.

As the first approximation model-free algorithm, over the next subsection, we analyze DQN, one of the most widely used algorithms in RL.

3.2.1 Deep Q-Networks (DQN)

The **DQN** algorithm integrates Q-learning [120] with NNs (see Fig. **??** for a depiction of a fully-connected neural network) to learn control policies directly from raw pixel inputs. It employs Convolutional Neural Networks (CNNs) to process these inputs, along with an experience replay mechanism that helps stabilize learning by breaking correlations between consecutive experiences. The target network, which is updated less frequently, further stabilizes the training process. DQN demonstrated state-of-the-art performance on various Atari 2600 games, outperforming previous methods and, in some cases, human experts, all while using a consistent network architecture and hyperparameters across games [121].

DQN integrates the Bellman equation with deep learning methods, including the loss function and gradient descent, to optimize and determine the optimal policy, as illustrated in the following:

Loss Function

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim D}\left[(y_i - Q(s,a;\theta_i))^2\right] \tag{3.1}$$

where

$$y_i = r + \gamma \max_{a'} Q(s',a';\theta^-) \tag{3.2}$$

Gradient Descent Step

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim D}\Big[\big(r + \gamma \max_{a'} Q(s',a';\theta^-) - Q(s,a;\theta_i)\big)\nabla_{\theta_i} Q(s,a;\theta_i)\Big] \tag{3.3}$$

Alg. 15 details the complete Deep Q-learning algorithm, regardless of using CNNs, as per the first paper. This algorithm is a generalization to have a general overview of the algorithm.

Figure 3.1: The architecture of the Deep Q-Network (DQN), as described in [121], employs a deep CNN to approximate the value function. It processes game screenshots from the Atari 2600 as direct input, producing the Q-values for all possible actions as its output [122].

Having established a solid foundation of the algorithm, we now turn to an analysis of the papers in the literature. In [123], authors explored the use of DQN for controlling Heating, Ventilation, and Air Conditioning (HVAC) systems in an energy-efficient manner without relying on simulation models. The authors applied DQN to a reference office building, simulating its performance with an EnergyPlus model, aiming to minimize energy use while maintaining indoor CO2 concentrations below 1,000 ppm. One of the key strengths of this study was its demonstration of how DQN could improve HVAC control by learning from previous actions, states, and rewards, thus offering a model-free optimization approach. The paper reported a significant reduction in total energy usage (15.7%) compared to baseline operations, highlighting the potential of DQN to enhance energy efficiency in buildings. The research also addressed the complexity and interconnectivity of HVAC systems, providing a practical solution to a traditionally challenging control problem. However, the paper also had some limitations. The reliance on simulation data for training and testing the DQN might not have fully captured the intricacies and variations of real-world scenarios. Additionally, the study focused on a specific type of building and HVAC setup, which might have limited the generalizability of the results to other building types or climates. Furthermore, the study did not delve deeply into the potential challenges of implementing such a system in practice, such as the need for extensive data collection and processing capabilities.

An innovative application of DQN to localize brain tumors in MRI images was introduced in [124]. The key strength of this study lies in its ability to generalize tumor localization with limited training data, addressing significant limitations of supervised DL, which often required large annotated datasets and struggled with generalization. The authors demonstrated that the DQN approach achieved 70% accuracy on a testing set with just 30 training images, significantly outperforming the supervised DL method that showed

Algorithm 15 Deep Q-learning [121]

Initialize replay memory $\mathcal{D}$ to capacity N and action-value function Q with random weights
for episode $= 1$ to M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1$ to T **do**
    With probability ϵ, select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

only 11% accuracy due to over-fitting. This showcased the robustness of RL in handling small datasets and its potential for broader application in medical imaging. The use of a grid-world environment to define state-action spaces and a well-designed reward system further strengthened the methodology. A notable weakness, however, was the limitation to two-dimensional image slices, which might not have fully captured the complexities of three-dimensional medical imaging. Future work should address this by extending the approach to 3D volumes and exploring more sophisticated techniques to improve stability and accuracy.

Authors in [125] presented a routing algorithm for enhancing the sustainability of Rechargeable Wireless Sensor Networks. The authors proposed an adaptive dual-mode routing approach that integrated multi-hop routing with direct upload routing, optimized through DQN. The strength of this paper lies in its innovative use of RL to dynamically adjust the routing mode based on the life expectancy of nodes, significantly improving the network's energy efficiency and lifespan. The simulation results demonstrated that the proposed algorithm achieved a correct routing mode selection rate of 95% with limited network state information, showcasing its practical applicability and robustness. However, the paper did not fully address potential real-world challenges, such as the computational overhead of implementing DQN in resource-constrained sensor nodes and the impact of

network topology changes over time. Additionally, while the simulations showed promising results, practical deployment and testing in diverse environments would have been necessary to validate the generalizability of the approach.

Researchers in [126] explored the application of DQN optimized using Particle Swarm Optimization (PSO) for resource allocation in fog computing environments, specifically tailored for healthcare applications. A notable strength of this study was its innovative combination of DQN and PSO, which effectively balanced resource allocation by reducing makespan and improving both average resource utilization and load-balancing levels. The methodology leveraged real-time data to dynamically adjust resources, showcasing a practical application of DQN in a critical domain. However, the paper could have benefited from a more detailed discussion on the scalability of the proposed system and its performance under varying network conditions. Additionally, while the results were promising, they were primarily based on simulations, which might not have fully captured the complexities and unpredictability of real-world fog environments. Further validation in practical deployments would have been necessary to fully ascertain the efficacy of the proposed approach.

An adaptive power management strategy for Parallel Plug-in Hybrid EVs (PHEVs) using DQN was investigated in [127]. The strength of this paper lies in its practical application of DQN for real-time power distribution in PHEVs. The approach considered continuous state variables such as battery State of Charge (SOC), required power, vehicle speed, and remaining distance ratio, which enhanced the model's adaptability and precision in dynamic driving conditions. The DQN model successfully minimized fuel consumption while maintaining battery SOC within desired limits, showing a 6% increase in fuel consumption compared to DP which was globally optimal but computationally impractical for real-time applications. However, the research was primarily based on simulations using the FTP-72 driving cycle, which might not have fully captured the variability of real-world driving conditions. Additionally, the study focused on a specific type of PHEV and driving scenario, which might have limited the generalizability of the results. Further real-world testing and validation across different vehicle models and driving conditions were necessary to establish the robustness and practical applicability of the proposed strategy.

Authors in [128] explored the application of DQN to create an AI for a visual fighting game. A significant strength of this study was its innovative reduction of the action space from 41 to 11 actions, which simplified the training process and enhanced the model's performance. The DQN architecture included convolutional and fully connected layers optimized for handling sequential frame inputs, effectively learning to perform complex combinations in a dynamic, competitive environment. However, a notable limitation was the reliance on a static opponent (None agent) during training, which might not have

fully captured the complexities and adaptive behaviors of actual gameplay against diverse opponents. Additionally, the experiments were conducted in a controlled environment with specific hardware, potentially limiting the generalizability of the results to other setups or real-world gaming scenarios. Further work should have focused on testing against more dynamic and varied opponents to evaluate the robustness and adaptability of the AI.

Authors in [129] presented a novel path-planning approach using an enhanced DQN combined with dense network structures. The strength of this work was its innovative policy of leveraging both depth and breadth of experience during different learning stages, which significantly accelerated the learning process. The introduction of a value evaluation network helped the model quickly grasp environmental rules, while the parallel exploration structure improved the accuracy by expanding the experience pool. The use of dense connections further enhanced feature propagation and reuse, contributing to improved learning efficiency and path planning success. However, the primary limitation was that the experiments were conducted in a controlled grid environment with specific sizes (5x5 and 8x8), which might not have fully represented the complexities of real-world scenarios. Additionally, the reliance on a fixed maximum number of steps could potentially have led to suboptimal policy evaluations in dynamic and larger environments. Future work should focus on validating this approach in more diverse and scalable settings to assess its generalizability and robustness.

The use of DQN for the real-time control of ESS co-located with renewable energy generators was discussed in [130]. A key strength of this work was its model-free approach, which did not rely on distributional assumptions for renewable energy generation or real-time prices. This flexibility allowed the DQN to learn optimal policies directly from interaction with the environment. The simulation results demonstrated that the DQN-based control policy achieved near-optimal performance, effectively balancing energy storage management tasks like charging and discharging without violating operational constraints. However, the primary limitation was the reliance on simulated data for both training and evaluation, which might not have captured all the complexities of real-world energy systems. Additionally, the approach assumed the availability of significant computational resources, which might not have been feasible for all consumers. Future work should have focused on real-world implementations and considered the computational constraints of practical deployment environments.

Study [131] presented a novel approach to countering intelligent Unmanned Aerial Vehicle (UAV) jamming attacks using a Stackelberg dynamic game framework. The UAV jammer, acting as the leader, used Deep Recurrent Q-Networks (DRQN) to optimize its jamming trajectory, while ground users, as followers, employed DQN to find optimal communication trajectories to evade the jamming. The strength of this paper was its compre-

hensive modeling of the UAV jamming problem using DRQN and DQN, which effectively handled the dynamic and partially observable nature of the environment. The approach proved effective in simulations, showing that both the UAV jammer and ground users could achieve optimal trajectories that maximized their respective long-term cumulative rewards. The Stackelberg equilibrium ensured that the proposed strategies were stable and effective in a competitive environment. However, the primary limitation was the complexity and computational demands of implementing DRQN and DQN in real-time scenarios, which might have been challenging in practical deployments. Additionally, the simulations were based on specific scenarios and parameters, which might not have fully captured the variability of real-world environments. Further validation through real-world experiments and a more extensive range of scenarios would have been necessary to confirm the robustness and scalability of the proposed approach.

In the following paragraphs, we examine Double Deep Q-Networks (DDQN), an extension of DQN designed to overcome the overestimation bias seen in Q-learning.

3.2.2 Double Deep Q-Networks (DDQN)

The **DDQN** algorithm is an extension of DQN, specifically designed to mitigate the overestimation bias found in Q-learning. It does so by separating the process of action selection from the evaluation of the Q-value, resulting in a more accurate estimation of action values [132, 133].

The key contributions of **DDQN** can be summarized as follows:

- **Overestimation Reduction**: By employing two distinct networks to select and evaluate actions, **DDQN** reduces the overestimation bias that is common in standard DQN.
- **Improved Stability and Performance**: The decoupling mechanism enhances both the stability and performance of the learning process, particularly in more complex environments.

The double Q-learning update rule is:

$$y = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-) \tag{3.4}$$

where θ are the parameters of the online network and θ^- are the parameters of the target network. Alg. 16, provides a general overview of the DDQN algorithm.

Algorithm 16 DDQN [134]

Initialize: Online network parameters θ, Target network parameters θ^-, Replay buffer $\mathcal{D}$, Exploration rate ϵ, and Discount factor γ
for each episode **do**
  while (not done) **do**
    Observe state s
    Select action a based on ϵ-greedy policy:
    $a \leftarrow \begin{cases} \text{random action} & \text{probability } \epsilon \\ \arg\max_a Q(s,a;\theta) & \text{probability } 1-\epsilon \end{cases}$
    Execute a, observe r and s'
    Store transition (s, a, r, s') in replay buffer $\mathcal{D}$
    Sample a mini-batch of transitions from $\mathcal{D}$
    Compute target:

$$y = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-)$$

    Update online network by minimizing loss:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}} \left[(y - Q(s,a;\theta))^2 \right]$$

    if Every C steps **then**
      Update the target network:

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

    end if
  end while
end for

Now, let us examine the papers in the literature that have utilized DDQN.

Authors in [135] explored the application of DDQN for optimizing energy efficiency in Cloud Radio Access Networks (CRAN). A key strength of this work was its innovative use of DDQN to address the Q-value overestimation issue inherent in traditional DQN methods. By separating the action selection and evaluation processes, the DDQN framework achieved more accurate and stable policy learning. The simulation results showed that DDQN significantly enhanced energy efficiency compared to baseline solutions and standard DQN approaches, demonstrating up to 22% power savings and a 20% improvement in EE. However, the study's reliance on simulated environments and predefined parameters

might not have fully reflected the complexities of real-world CRAN scenarios. The scalability and adaptability of the proposed method to diverse network conditions and dynamic user demands needed further validation through practical implementations. Additionally, while the study provided a robust theoretical framework, the practical deployment challenges and computational overheads associated with DDQN in live networks required more in-depth exploration.

The application of DDQN to object detection tasks was investigated in [136]. One of the key strengths of this work was its innovative use of DDQN to enhance the accuracy and efficiency of object detection. By decoupling action selection and evaluation using two separate Q-networks, the DDQN approach addressed the overestimation problem associated with traditional DQN methods. This led to higher precision and recall rates, as demonstrated in their experiments. The method proved efficient, requiring fewer steps to detect objects and showing strong adaptability to different environments, including person detection scenarios. However, the study relied heavily on specific datasets and controlled environments, which might have limited the generalizability of the results. Real-world applications could have presented more complex scenarios that might have challenged the robustness of the proposed method. Further validation in diverse and uncontrolled settings would have helped ascertain the practical applicability and scalability of the DDQN-based object detection framework.

Authors in [137] presented an enhanced path-tracking approach for robotic vehicles using DDQN. A key strength of this work was its innovative application of DDQN for both path smoothing and tracking, which significantly reduced overshoot and settling time compared to traditional methods like pure pursuit control. The proposed method demonstrated superior performance in navigating paths with sharp turns, making it particularly suitable for agricultural applications where precise path following was critical. The use of a simulation environment for training and subsequent testing on a real rover enhanced the robustness of the developed algorithm. However, the reliance on a specific rover model and controlled environments might have limited the generalizability of the results. The study could have benefited from further validation in diverse real-world conditions and with different types of robotic vehicles to fully establish the robustness and adaptability of the proposed method. Additionally, the computational demands of implementing DDQN in real-time applications might have posed practical challenges that needed addressing.

A novel method for improving tactical decision-making in autonomous driving using DDQN enhanced with spatial and channel attention mechanisms was introduced in [138]. The strength of this paper lies in its innovative use of a hierarchical control structure that integrated spatial and channel attention modules to better encode the relative importance of different surrounding vehicles and their features. This approach allowed for

more accurate and efficient decision-making, as evidenced by the significant improvement in safety rates (54%) and average exploration distance (30%) in simulated environments. The combination of the algorithm with double attention enhanced the agent's ability to make intelligent and safe tactical decisions, outperforming baseline models in terms of both reward and capability metrics. However, the reliance on simulation-based testing limited the assessment of the model's performance in real-world driving scenarios, which might have presented additional complexities and unpredictabilities. The computational overhead associated with the attention modules and the utilized algorithm could also have posed challenges for real-time implementation in autonomous vehicles. Further validation in diverse and realistic environments would have been essential to confirm the robustness and scalability of this approach.

Study [139] explored a novel approach to solving the distributed heterogeneous hybrid flow-shop scheduling problem with multiple priorities of jobs using a DDQN-based co-evolutionary algorithm. A key strength of this work was its comprehensive framework that integrated global and local searches to balance computational resources effectively. The proposed DDQN-based co-evolutionary algorithm showed significant improvements in minimizing total weighted tardiness and total energy consumption by efficiently selecting operators and accelerating convergence. The numerical experiments and comparisons with state-of-the-art algorithms demonstrated the superior performance of the proposed method, particularly in handling real-world scenarios and large-scale instances. However, the study primarily relied on simulations and controlled experiments, which might not have fully captured the complexities and variability of actual manufacturing environments. Additionally, the computational overhead associated with training the DDQN model could have posed challenges for real-time applications. Further validation through real-world implementations and exploring dynamic events such as new job inserts and due date changes would have been necessary to fully establish the robustness and practical applicability of the proposed method.

A method for autonomous mobile robot navigation and collision avoidance using DDQN was developed in [140]. A significant strength of this work was its innovative use of DDQN to reduce reaction delay and improve training efficiency. The proposed method demonstrated superior performance in navigating robots to target positions without collisions, even in multi-obstacle scenarios. By employing a Kinect2 depth camera for obstacle detection and leveraging a well-designed reward function, the method achieved quick convergence and effective path planning, as evidenced by both simulation and real-world experiments on the Qbot2 robot platform. However, the reliance on a controlled laboratory environment and specific hardware (e.g., Optitrack system for positioning) might have limited the generalizability of the results. Real-world applications could have introduced additional

complexities and variabilities that the study did not address. Further validation in diverse and uncontrolled environments, along with a discussion on the computational requirements and scalability of the approach, would have been necessary to fully establish its practical applicability.

Authors in [141] implemented an RL-based method for autonomous vehicle decision-making in overtaking scenarios with oncoming traffic. The study leveraged DDQN to manage both longitudinal speed and lane-changing decisions. A notable strength of this research was the use of DDQN with prioritized experience replay, which accelerated policy convergence and enhanced decision-making precision. The simulation results in SUMO showed that the proposed method improved average speed, reduced time spent in the opposite lane, and lowered the overall overtaking duration compared to traditional methods, such as SUMO's default lane-change model. The RL-based approach demonstrated a high collision-free rate (98.5%) and effectively mimicked human decision-making behavior, showcasing significant improvements in safety and efficiency. However, the reliance on simulated environments and specific scenarios might have limited the generalizability of the results to real-world applications. The study assumed complete observability of the state space, which might not have been realistic in actual driving conditions where sensor imperfections and uncertainties were common. Future work should focus on addressing these limitations by exploring POMDPs and validating the approach in diverse, real-world scenarios.

A DDQN-based control method for obstacle avoidance in agricultural robots was introduced in[142]. A key strength of this work was its innovative use of DDQN to handle the complexities of dynamic obstacle avoidance in a structured farmland environment. The proposed method effectively integrated real-time data from sensors and used a neural network to decide the optimal actions, leading to significant improvements in space utilization and time efficiency compared to traditional risk index-based methods. The study reported high success rates in obstacle avoidance (98-99%) and demonstrated the model's robustness through extensive simulations and field experiments. However, the reliance on predefined paths and the assumption of only dynamic obstacles appearing on these paths might have limited the approach's flexibility in more complex or unstructured environments. Additionally, the computational demands of the DDQN framework might have posed challenges for deployment in the resource-constrained settings typical of agricultural machinery. Further validation in diverse real-world scenarios and exploration of more scalable solutions could have enhanced the practical applicability of this method.

A DDQN-based algorithm for global path planning of amphibious Unmanned Surface Vehicles (USVs) was studied in [143]. A major strength of this paper was its innovative use of DDQN for handling the complex path-planning requirements of amphibious USVs,

which must navigate both water and air environments. The integration of electronic nautical charts and elevation maps to build a detailed 3D simulation environment enhanced the realism and accuracy of the path planning. The proposed method effectively balanced multiple objectives, such as minimizing travel time and energy consumption, making it suitable for diverse scenarios including emergency rescue and long-distance cruising. However, the study primarily relied on simulated environments and predefined scenarios, which might not have fully captured the complexities of real-world applications. The computational demands of the DDQN framework could also have posed challenges for real-time implementation, especially in dynamic environments where quick decision-making was crucial. Further validation through practical deployments and testing in various real-world conditions would have been necessary to fully assess the robustness and scalability of the proposed approach.

Authors in [144] presented a Hand-Over (HO) strategy for 5G networks. The proposed Intelligent Dual Active Protocol Stack (I-DAPS) HO used DDQN to enhance the reliability and throughput of handovers by predicting and avoiding radio link failures. A key strength of this paper was its innovative approach to leveraging DDQN for dynamic and proactive handover decisions in highly variable mmWave environments. The use of a learning-based framework allowed the system to make informed decisions based on historical signal data, thereby significantly reducing HO failure rates and achieving zero-millisecond mobility interruption time. The simulation results demonstrated substantial improvements in throughput and reliability over conventional handover schemes like conditional HO and baseline dual active protocol stack HO. However, the study's primary reliance on simulations and specific urban scenarios might have limited the generalizability of the findings. Real-world implementations might have faced additional challenges, such as varying environmental conditions and hardware limitations that were not fully addressed in the simulations. Further validation in diverse real-world environments would have been necessary to confirm the practical applicability and robustness of the proposed I-DAPS HO scheme.

Authors in [145] implemented an advanced vehicle speed control system using DDQN. The approach integrated high-dimensional video data and low-dimensional sensor data to construct a comprehensive driving environment, allowing the system to mimic human-like driving behaviors. A key strength of this work was its effective use of DDQN to address the instability issues found in Q-learning, resulting in more accurate value estimates and higher policy quality. The use of naturalistic driving data from the Shanghai naturalistic driving study enhanced the model's realism and applicability. The system demonstrated substantial improvements in both value accuracy and policy performance, achieving a score that was 271.73% higher than that of DQN. However, the reliance on pre-recorded driv-

ing data and controlled environments might have limited the generalizability of the results. Real-world driving conditions could have been significantly more variable, and the system's performance in such dynamic environments needed further validation. Additionally, the computational demands of DDQN might have posed challenges for real-time implementation in autonomous vehicles, necessitating further optimization for practical deployment.

Authors in [146] addressed the dynamic multiple-channel access problem in IoT networks with energy harvesting devices using the DDQN approach. The key strength of this study lies in its innovative application of DDQN to manage scheduling policies in POMDP. By converting the partial observations of scheduled nodes into belief states for all nodes, the proposed method effectively reduced energy costs and extended the network's lifetime. The simulation results demonstrated that DDQN outperformed other RL algorithms, including Q-learning and DQN, in terms of average reward per time slot. However, the paper's reliance on simulations and specific parameters might have limited its applicability to real-world IoT environments. The assumptions regarding the Poisson process for energy arrival and the fixed transmission energy threshold might not have fully captured the complexities of practical deployments. Further validation through real-world experiments and consideration of diverse environmental conditions would have been necessary to fully establish the robustness and scalability of the proposed scheduling approach.

In the next subsection, we introduce another variant of DQN that utilizes the Dueling architecture.

3.2.3 Dueling Deep Q-Networks (Dueling DQN)

The **Dueling DQN** algorithm introduces a novel NNS architecture that separates the representation of state values and advantages, enhancing both learning efficiency and performance. This structure allows the network to estimate the value of each state more accurately, leading to improved policy evaluation and greater stability. The dueling architecture divides the Q-network into two streams: one estimates the state value function, and the other estimates the advantage function, which is then combined to generate the Q-values. By separately estimating the state value and advantage, the dueling network provides more informative gradients, resulting in more efficient learning. This architecture has shown improved performance in a variety of RL tasks, especially in environments where many actions have similar values [147].

The dueling network update rule is given as:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|A|} \sum_{a'} A(s, a'; \theta, \alpha) \right) \tag{3.5}$$

where θ are the parameters of the shared network, α are the parameters of the advantage stream, and β are the parameters of the value stream.

By separately estimating the state value and advantage, the dueling network offers more robust estimates of state values. This decoupling mechanism improves stability and overall performance, particularly in environments where many actions have similar values. Alg. 17 provides a comprehensive overview of the Dueling DQN algorithm.

In the following paragraphs, several research studies are analyzed in detail to gain a deeper understanding of the Dueling DQN algorithm.

A DRL-based framework using Dueling Double Deep Q-Networks with Prioritized Replay (DDDQNPR) to solve adaptive job shop scheduling problems was developed in [148]. The primary strength of this study lies in its innovative integration of DDDQNPR with a disjunctive graph model, which transformed scheduling into a sequential decision-making process. This approach enabled the model to adapt to dynamic environments and learn optimal scheduling policies through offline training. The method outperformed traditional heuristic rules and genetic algorithms in both static and dynamic scheduling scenarios, demonstrating substantial improvements in scheduling performance and generalization. However, the reliance on predefined benchmarks and controlled experiments may have limited the real-world applicability of the results. Additionally, the complexity and computational demands of the DDDQNPR framework could present challenges for real-time implementation in large-scale production environments. Future work should focus on validating the approach in more diverse real-world settings and exploring more efficient reward functions and neural network structures to improve the scalability and robustness of the method.

Managing Device-to-Device (D2D) communication using Dueling DQN architecture was proposed by authors in [149]. A key strength of this paper was its effective utilization of the Dueling DQN architecture to autonomously determine transmission decisions in D2D networks without relying on centralized infrastructure. The approach leveraged easily obtainable Channel State Information (CSI), allowing each D2D transmitter to train its neural network independently. The proposed method successfully mitigated co-channel interference, demonstrating near-optimal sum rates in low signal-to-noise ratio environments and outperforming traditional schemes like no control, opportunistic, and Suboptimal in terms of efficiency and complexity reduction. However, the study's reliance on simulation data and specific channel models might have limited the generalizability of the results to real-world D2D networks. The approach assumed ideal conditions, such as perfect CSI and zero delay in TDD-based full duplex communications, which might not have held in practical scenarios. Future work should have focused on validating the method in diverse real-world environments and addressing potential implementation challenges, such as real-

time computational requirements and varying network conditions.

Authors explored an advanced power allocation algorithm in edge computing environments using Dueling DQN in [150]. A significant strength of this study was its innovative use of Dueling DQN, which separated the value and advantage functions within the Q-network. This architecture enhanced the accuracy and efficiency of power allocation by focusing on state-value estimation and advantage learning. The proposed algorithm outperformed several baseline methods, including traditional DQN, fractional programming, and weighted minimum mean square error, in both simulated scenarios and different user densities. The results demonstrated substantial improvements in average downlink rates and reduced computational overhead, highlighting the algorithm's robustness and efficiency in dynamic network conditions. However, the paper relied on simulations with specific parameters and controlled environments, which might have limited its applicability to real-world scenarios. The assumptions regarding perfect synchronization and the static nature of edge servers might not have fully captured the complexities of practical deployments. Further real-world testing and validation were needed to confirm the scalability and practical implementation of the proposed Dueling DQN approach in diverse and dynamic edge computing environments.

A hybrid approach combining Dueling DQN and Double Elite Co-Evolutionary Genetic Algorithm (DECGA) for parameter estimation in variogram models used in geo-statistics and environmental engineering was implemented in [151]. A notable strength of this study was its innovative integration of RL with evolutionary algorithms. The Dueling DQN enhanced the parameter setting of the genetic algorithm, improving the convergence speed and avoiding premature convergence. The DECGA was effective in finding global optimal solutions by maintaining a diverse population. This combined approach significantly improved the accuracy of parameter estimation in both single and nested variogram models, as demonstrated by the experimental results on heavy metal concentration datasets, showing lower residual sum of squares values compared to traditional methods. However, the reliance on controlled datasets and predefined parameters might have limited the method's generalizability to more complex, real-world scenarios. The computational complexity introduced by combining Dueling DQN with DECGA could also have posed challenges for practical implementation, especially in scenarios requiring real-time parameter estimation. Future work should focus on validating the approach in diverse real-world environments and optimizing the computational efficiency for broader applicability.

Authors in [153] presented a method for optimizing UAV flight paths in IoT sensor networks using Dueling DQN. The goal was to balance energy consumption and data delay. A significant strength of the study was its effective use of Dueling DQN to manage UAV paths, enhancing both energy efficiency and network performance. The approach utilized

Algorithm 17 Dueling DQN [152]

Input: Learning rate lr, batch size $batch$, discount factor γ, exploration rate ϵ, decay rate $decay$, target network update frequency $freq$, episodes
Output: Trained Dueling Q-Network
Initialize online network weights θ
Initialize target network $\theta^- = \theta$
Initialize replay memory $\mathcal{D}$
for episode $e = 1$ to $episodes$ **do**
    Initialize state s
    while state s is not terminal **do**
        Select action a_t using ϵ-greedy policy
        Execute a_t, observe r_t, s_{t+1}
        Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
        Sample minibatch from $\mathcal{D}$
        Compute targets y_j
        Update θ by minimizing loss $L(\theta)$
        if update step mod $freq = 0$ **then**
            Update target network $\theta^- \leftarrow \theta$
        end if
        $s \leftarrow s_{t+1}$
        Decay ϵ
    end while
end for

a grid-based method to model network states, reducing computational complexity. Simulation results showed notable improvements in power consumption and data transmission delays compared to existing methods. However, the reliance on simulated environments might have limited the applicability of the results to real-world scenarios, where sensor data and network conditions could have been more variable. Further validation through practical implementations and testing in diverse environments would have been essential to confirm the robustness and scalability of the proposed method. Additionally, the computational demands of Dueling DQN might have posed challenges for real-time applications in resource-constrained IoT networks.

An innovative method for optimizing the Age of Information (AoI) in vehicular fog systems using Dueling DQN was presented in [154]. A major strength of this work was its holistic approach to AoI optimization, considering both the transmission time from the information source to the monitor and from the monitor to the destination. The Dueling

DQN algorithm effectively minimized the end-to-end AoI by leveraging edge computing and vehicular fog nodes to handle real-time data more efficiently. The proposed method demonstrated significant improvements in system performance and information freshness compared to DQN and analytical methods. However, the reliance on simulations to validate the approach might have limited the generalizability of the findings to real-world applications. The assumptions made regarding the processing capabilities and network conditions might not have fully captured the complexities of practical implementations. Further research involving real-world testing and validation in diverse environments was necessary to fully establish the robustness and scalability of the proposed method.

The authors in [155] presented an enhanced Dueling DQN method for improving the autonomous capabilities of UAVs in terms of obstacle avoidance and target tracking. One of the key strengths of this study was the effective integration of improved Dueling DQN with a target tracking mechanism, which allowed the UAV to make more precise and timely decisions. The improved algorithm showed superior performance in simulation tests, with significant improvements in obstacle avoidance accuracy and target tracking efficiency. The authors provided a comprehensive analysis of the algorithm's ability to maintain stability and adaptability in dynamic environments, showcasing its practical potential for real-world UAV applications. However, the study's dependence on simulation environments and controlled scenarios might not have fully captured the complexities and unpredictability of real-world applications. The effectiveness of the improved Dueling DQN algorithm in diverse and unstructured environments needed further validation through real-world testing. Additionally, the computational demands of the improved algorithm could have posed challenges for real-time processing and decision-making on resource-constrained UAV platforms. Future work should have addressed these aspects to establish the practicality and scalability of the proposed approach fully.

An advanced path planning approach for Unmanned Surface Vessels (USVs) using Dueling DQN enhanced with a tree sampling mechanism was implemented in [156]. A key strength of this work was the innovative combination of Dueling DQN with a tree-based priority sampling mechanism, which significantly improved the convergence speed and efficiency of the learning process. The approach leveraged the decomposition of the value function into state-value and advantage functions, enhancing the policy evaluation and decision-making accuracy. The simulation results demonstrated that the proposed method achieved faster convergence and more effective obstacle avoidance and path planning compared to DQN algorithms. Specifically, the Dueling DQN algorithm showed improved performance in terms of the number of steps and time required to reach the target across various test environments. However, the study's reliance on simulated static environments might have limited the generalizability of the results to real-world scenarios where the USVs

would have encountered dynamic and unpredictable conditions. The paper also did not address the computational complexity introduced by the tree sampling mechanism, which could have posed challenges for real-time implementation on resource-constrained USVs. Future work should focus on validating the approach in dynamic real-world environments and exploring ways to optimize the computational efficiency for practical applications. A summary of the analyzed papers is given in Table **??**. In the next subsection, another part of tabular model-based algorithms, model-based planning, is introduced.

3.2.4 Advanced Tabular Model-based Methods

In this subsection, we explore the second part of tabular model-based methods. Key features of model-based algorithms include model representation, which uses transition probabilities $P(s'|s,a)$ and reward functions $R(s,a)$ to describe state transitions and rewards; planning and policy evaluation, which iteratively approximates the value function $V(s)$ using the Bellman equation until convergence; and value iteration, which also refines $V(s)$ iteratively using the Bellman equation to determine the optimal policy by evaluating future states and actions [3].

After reviewing the first part of tabular model-based algorithms (i.e., DP approaches), we now shift our focus to the second part, model-based planning methods.

3.2.5 Model-based Planning

Model-based planning in RL refers to the approach where the agent constructs and uses a model of the environment to plan its actions and make decisions. This model can either be learned from interactions with the environment or be predefined if the environment's dynamics are already known. Model-based methods can be more sample-efficient as they use learned models to simulate various scenarios, enabling the agent to gather insights without needing to interact directly with the environment every time. Model-based planning helps the agent effectively balance exploration (trying new actions) and exploitation (using known effective actions). By simulating different actions and their potential outcomes, the agent can make more informed decisions, optimizing its behavior. The learned model allows the agent to easily adapt to environmental changes. Through model updates and replanning, the agent can dynamically respond to new situations, maintaining high performance even in changing environments [3, 30, 157].

There are three primary approaches in model-based planning: Monte Carlo Tree Search (MCTS), prioritized sweeping, and Dyna-Q. We begin by analyzing MCTS in the following paragraphs.

Figure 3.2: The basic MCTS process [2]

Monte Carlo Tree Search (MCTS)

MCTS is a powerful algorithm designed to tackle sequential decision-making problems under uncertainty. It has found wide application across various domains, particularly in-game AI, where it has achieved notable success. MCTS blends the precision of tree search with the generality of random sampling (Monte Carlo simulation). The core process of MCTS is conceptually simple, as illustrated in Fig. 3.2. This technique has been crucial in the development of AI systems such as AlphaGo and AlphaZero, which have demonstrated superhuman performance in games like Go, chess, and shogi. The term "MCTS" was initially coined by [158] when applied to the Go-playing program Crazy Stone. Before this, game-playing algorithms primarily used deterministic search techniques, like those employed in chess. The vast state spaces of games like Go presented significant challenges, necessitating a more intelligent and efficient search strategy, which resulted in the adoption of MCTS.

MCTS operates by incrementally building a search tree in an asymmetric manner, as illustrated in Fig. 3.3. Each iteration of MCTS involves four primary steps [3]:

1. **Selection**: Starting from the root node, the algorithm recursively selects child nodes until it reaches a leaf node (lines 14-17). This process balances exploration and exploitation, typically using a UCB formula to prioritize moves with high estimated value and uncertainty.

Figure 3.3: One iteration of the general MCTS approach

2. **Expansion**: If the leaf node is not a terminal node, it is expanded by adding one or more child nodes (line 18). This step explores new, previously unconsidered moves or actions.

3. **Simulation**: From the expanded node, a simulation (or playout) is executed to the end of the game using a default policy, often random (lines 19-21). The simulation provides an estimated outcome that helps approximate the value of the moves from the expanded node.

4. **Backpropagation**: The result of the simulation is propagated back through the tree, updating the values of all nodes along the path (line 22). This refines the estimated value of moves based on the new data.

These steps are repeated until the computational budget (time or iterations) is exhausted. The move with the highest value at the root node is selected as the optimal move. Based on the overview in [159], Alg. 18 provides a comprehensive overview of MCTS.

Key Advantages of MCTS are domain independence (MCTS does not require domain-specific knowledge, only the rules of the game or problem.), scalability (It can handle very large state spaces by focusing computational effort on the most promising parts of the search tree.), and flexibility (It can be adapted to various types of problems, including those with stochastic elements) [158, 3, 160].

Algorithm 18 MCTS [159]

```
Input:
    PD: problem's dimension
    FG: fitness goal
    RL: roll-out length
Output:
    n*: the best solution found
    f: the maximum performance level reached
    t: the total number of roll-outs performed
Create root node n_r and add it to the tree
Make n_r the actual node n
n* ← null
f ← 0
t ← 0
while move_available(n) do
    while not is_leaf(n) do
        n ← best_child(n)
    end while
    n ← add_random_child_node(n)
    RL ← rand(PD, n)
    f, n' ← roll_out(n, RL)
    t ← t + 1
    back_propagate(n, f)
    if f > FG then
        n* ← n'
        break
    end if
    n ← n_r
end while
return n*, f, t
```

MCTS is a fundamental and crucial algorithm in the field of RL. Therefore, let us explore some studies that have applied this approach to solve various problems.

Study [161] presented the Multi-objective MCTS (MO-MCTS) algorithm, an extension of MCTS designed for multi-objective sequential decision-making. The key innovation lies in using the hyper-volume indicator to replace the UCB criterion, enabling the algorithm to handle multi-dimensional rewards and discover several Pareto-optimal policies within

a single tree. The MO-MCTS algorithm stood out for its ability to effectively manage multi-objective optimization by integrating the hyper-volume indicator, which provided a comprehensive measure of the solution set's quality. This approach allowed the algorithm to balance exploration and exploitation in multi-dimensional spaces, making it more efficient in discovering Pareto-optimal solutions compared to traditional scalarized RL methods. The use of the hyper-volume indicator as an action selection criterion ensured that the algorithm could capture a diverse set of optimal policies, addressing the limitations of linear-scalarization methods that failed in non-convex regions of the Pareto front. The experimental validation on the Deep Sea Treasure (DST) problem and grid scheduling tasks demonstrated that MO-MCTS achieved superior performance and scalability, matching or surpassing state-of-the-art non-RL-based methods despite higher computational costs. However, the reliance on accurate computation of the hyper-volume indicator introduced significant computational overhead, particularly in high-dimensional objective spaces. The complexity of maintaining and updating the Pareto archive could also pose challenges, especially as the number of objectives increases. While the algorithm showed robust performance in deterministic settings, its scalability and efficiency in highly stochastic or dynamic environments remained to be fully explored. The need for domain-specific knowledge to define the reference point and other hyper-volume-related parameters could also limit the algorithm's generalizability. Additionally, the method's computational intensity might hinder its real-time applicability, requiring further optimization or approximation techniques to reduce processing time.

A novel approach to Neural Architecture Search (NAS) using MCTS was introduced in [162]. The authors proposed a method that captured the dependencies among layers by modeling the search space as a Monte-Carlo Tree, enhancing the exploration-exploitation balance and efficiently storing intermediate results for better future decisions. The method was validated through experiments on the NAS-Bench-Macro benchmark and ImageNet dataset. The primary strength of this approach lies in its ability to incorporate dependencies between different layers during architecture search, which many existing NAS methods overlook. By utilizing MCTS, the proposed method effectively balanced exploration and exploitation, leading to more efficient architecture sampling. The use of a Monte-Carlo Tree allowed for the storage of intermediate results, significantly improving the search efficiency and reducing the need for redundant computations. The introduction of node communication and hierarchical node selection techniques further refined the evaluation of numerous nodes, ensuring more accurate and efficient search processes. Experimental results demonstrated that the proposed method achieved higher performance and search efficiency compared to state-of-the-art NAS methods, particularly on the NAS-Bench-Macro and ImageNet datasets. On the other hand, the reliance on the accurate modeling of the search space as a Monte-Carlo Tree introduced additional complexity, particularly in terms

of computational overhead and memory requirements. The method's performance heavily depended on the proper tuning of hyperparameters, such as the temperature term and reduction ratio, which could be challenging to optimize for different datasets and tasks. While the experiments showed promising results, further validation in more diverse and complex real-world scenarios was necessary to fully assess the scalability and robustness of the approach. The additional computational cost associated with maintaining and updating the Monte-Carlo Tree and performing hierarchical node selection could impact the method's applicability in real-time applications or resource-constrained environments.

Study [163] introduced multi-agent MCTS (MAMCTS), a novel extension of MCTS tailored for cooperative multi-agent systems. The primary innovation was the integration of difference evaluations, which significantly enhanced coordination strategies among agents. The performance of MAMCTS was demonstrated in a multi-agent path-planning domain called multi-agent Grid-world (MAG), showcasing substantial improvements over traditional reward evaluation methods. The MAMCTS approach effectively leveraged difference evaluations to prioritize actions that contributed positively to the overall system, leading to significant improvements in learning efficiency and coordination among agents. By combining MCTS with different rewards, the algorithm balanced exploration and exploitation, ensuring that agents could efficiently navigate the search space to find optimal policies. The experimental results in the 100x100 MAG environment demonstrated that MAMCTS outperformed both local and global reward methods, achieving up to 31.4% and 88.9% better performance, respectively. This superior performance was consistent across various agent and goal configurations, highlighting the scalability and robustness of the approach. The use of a structured search process and prioritized updates ensured that the algorithm could handle large-scale multi-agent environments effectively. The reliance on accurate computation and maintenance of difference evaluations introduced additional computational overhead, particularly in environments with a large number of agents and goals. The method's performance was sensitive to the accuracy of these evaluations, which might be challenging to maintain in highly dynamic or unpredictable environments. While the simulations provided strong evidence of the method's efficacy, further validation in more diverse real-world scenarios was necessary to fully assess its scalability and practical utility. The complexity of managing multiple agents and ensuring synchronized updates could also pose challenges, particularly in real-time applications where computational resources are limited. Additionally, the paper focused primarily on cooperative settings, and the applicability of MAMCTS to competitive or adversarial multi-agent environments remained an area for future exploration.

A MO-MCTS algorithm tailored for real-time games was developed in [164]. It focused on balancing multiple objectives simultaneously, leveraging the hyper-volume indicator to

replace the traditional UCB criterion. The algorithm was tested against single-objective MCTS and Non-dominated Sorting Genetic Algorithm (NSGA)-II, showcasing superior performance in benchmarks like DST and the multi-objective Physical Traveling Salesman Problem (MO-PTSP). The MO-MCTS algorithm excelled in handling multi-dimensional reward structures, efficiently balancing exploration and exploitation. By incorporating the hyper-volume indicator, the algorithm could discover a diverse set of Pareto-optimal solutions, effectively addressing the limitations of linear-scalarization methods in non-convex regions of the Pareto front. The empirical results on DST and MO-PTSP benchmarks demonstrated the algorithm's ability to converge to optimal solutions quickly, outperforming both single-objective MCTS and NSGA-II in terms of exploration efficiency and solution quality. The use of a weighted sum and Euclidean distance mechanisms for action selection further enhanced the adaptability of MO-MCTS to various game scenarios, providing a robust framework for real-time decision-making. Nonetheless, the reliance on the hyper-volume indicator introduced significant computational overhead, which could limit the algorithm's scalability in high-dimensional objective spaces. The need for maintaining a Pareto archive and computing the hyper-volume for action selection added to the complexity, potentially impacting real-time performance. While the algorithm showed strong results in the tested benchmarks, its applicability to more complex and dynamic real-time games required further validation. The computational intensity of the approach might hinder its practicality in resource-constrained environments, necessitating the exploration of optimization techniques to reduce processing time without compromising performance. Additionally, the paper primarily focused on deterministic settings, leaving the performance in stochastic or highly variable environments less explored.

Researchers in [165] explored hybrid algorithms that integrated MCTS with mini-max search to leverage the strategic strengths of MCTS and the tactical precision of minimax. The authors proposed three hybrid approaches: employing minimax during the selection/expansion phase, the rollout phase, and the backpropagation phase of MCTS. These hybrids aimed to address the weaknesses of MCTS in tactical situations by incorporating shallow minimax searches within the MCTS framework. The hybrid algorithms presented in the paper offered a promising combination of MCTS's ability to handle large search spaces with mini-max's tactical accuracy. By integrating shallow mini-max searches, the hybrids could better navigate shallow traps that MCTS might overlook, leading to more robust decision-making in games with high tactical demands. The experimental results in games like Connect-4 and Breakthrough demonstrated that these hybrid approaches could outperform standard MCTS, particularly in environments where tactical precision was crucial. The use of mini-max in the selection/expansion phase and the backpropagation phase significantly improved the ability to avoid blunders and recognize winning strategies early, enhancing the overall efficiency and effectiveness of the search process. However,

the inclusion of mini-max searches introduced additional computational overhead, which could slow down the overall search process, especially for deeper mini-max searches. The performance improvements were heavily dependent on the correct tuning of parameters, such as the depth of the mini-max search and the criteria for triggering these searches. While the hybrids showed improved performance in the tested games, their scalability and effectiveness in more complex and dynamic real-world scenarios remained to be fully validated. The reliance on game-specific characteristics, such as the presence of shallow traps, might limit the generalizability of the results. Further exploration was needed to assess the impact of these hybrids in a broader range of domains and under different conditions.

Authors in [166] introduced the Option-MCTS (O-MCTS) algorithm, which extended the MCTS by incorporating high-level action sequences, or "options," aimed at achieving specific subgoals. The proposed algorithm aimed to enhance general video game playing by utilizing higher-level planning, enabling it to perform well across a diverse set of games from the General Video Game AI competition. Additionally, the paper introduced Option Learning MCTS (OL-MCTS), which applied a progressive widening technique to focus exploration on the most promising options. The integration of options into MCTS was a significant advancement, allowing the algorithm to plan more efficiently by considering higher-level strategies. This higher abstraction level helped the algorithm deal with complex games that required achieving multiple subgoals. The use of options reduced the branching factor in the search tree, enabling deeper exploration within the same computational budget. The empirical results demonstrated that O-MCTS outperformed traditional MCTS in games requiring sequential subgoal achievements, such as collecting keys to open doors, showcasing its strength in strategic planning. The introduction of OL-MCTS further improved performance by learning which options were most effective, thus focusing the search on more promising parts of the game tree and improving efficiency. On the other hand, the reliance on predefined options and their proper tuning could be a limitation, as the performance of O-MCTS heavily depended on the quality and relevance of these options to the specific games being played. The initial computational overhead associated with constructing and managing a large set of options might impact the algorithm's performance, particularly in games with numerous sprites and complex dynamics. The progressive widening technique in OL-MCTS, while beneficial for focusing exploration, introduced additional complexity and overhead, potentially reducing real-time applicability. Further validation was needed to assess the scalability and robustness of these algorithms in a wider range of real-world game scenarios, where the diversity and unpredictability of game mechanics might present new challenges.

An extension of MCTS, incorporating heuristic evaluations through implicit mini-max backups, was investigated in [167]. The approach aimed to combine the strengths of MCTS

and mini-max search to improve decision-making in strategic games by maintaining separate estimations of win rates and heuristic evaluations and using these to guide simulations. The integration of implicit minimax backups within MCTS significantly enhanced the quality of simulations by leveraging heuristic evaluations to inform decision-making. This hybrid approach addressed the limitations of pure MCTS in domains where tactical precision was crucial, effectively balancing strategic exploration with tactical accuracy. By maintaining separate values for win rates and heuristic evaluations, the algorithm could better navigate complex game states, leading to stronger play performance. The empirical results in games like Kalah, breakthrough, and Lines of Action demonstrated substantial improvements over standard MCTS, validating the effectiveness of implicit mini-max backups in diverse strategic environments. The method also showed robust performance across different parameter settings, highlighting its adaptability and potential for broader application in game-playing AI. However, the reliance on accurate heuristic evaluations introduced complexity in environments where such evaluations were not readily available or were difficult to compute. The additional computational overhead associated with maintaining and updating separate value estimates might impact the algorithm's efficiency, particularly in real-time applications. While the approach showed significant improvements in specific games, further validation was necessary to assess its scalability and robustness in more complex and dynamic scenarios. The dependence on domain-specific knowledge for heuristic evaluations might limit the generalizability of the method to a wider range of applications. Additionally, the complexity of tuning the parameter that weighted the influence of heuristic evaluations could pose challenges in optimizing the algorithm for different environments.

Authors in [168] explored the application of MCTS to the game of Lines of Action (LoA), a two-person zero-sum game known for its tactical depth and moderate branching factor. The authors proposed several enhancements to standard MCTS to handle the tactical complexities of LoA, including game-theoretical value proving, domain-specific simulation strategies, and effective use of progressive bias. The key strength of this paper lies in its innovative enhancements to MCTS, enabling it to handle the tactical and progression properties of LoA effectively. By incorporating game-theoretical value proving, the algorithm could identify and propagate winning and losing positions more efficiently, reducing the computational burden of extensive simulations. The use of domain-specific simulation strategies significantly improved the quality of the simulations, leading to more accurate evaluations and better overall performance. The empirical results demonstrated that the enhanced MCTS variant outperformed the world's best $\alpha\beta$-based LoA program, marking a significant milestone for MCTS in handling highly tactical games. The detailed analysis and systematic approach to integrating domain knowledge into MCTS provided a robust framework for applying MCTS to other complex games. Despite its advantages, the

approach introduced additional computational overhead due to the need for maintaining and updating multiple enhancements, such as the progressive bias and game-theoretical value proving. The reliance on domain-specific knowledge for simulation strategies and the progressive bias limited the generalizability of the method to other games without similar properties. While the algorithm performed well in the controlled environment of LoA, its scalability and robustness in more dynamic and less structured environments remained to be fully explored. The complexity of tuning various parameters, such as the progressive bias coefficient and the simulation cutoff threshold, could also pose challenges, particularly for practitioners without deep domain knowledge. Further validation in diverse real-world scenarios was necessary to assess the practical applicability and long-term benefits of the proposed enhancements.

Authors in [169] explored the application of MCTS to the popular collectible card game "Hearthstone: Heroes of Warcraft." Given the game's complexity, uncertainty, and hidden information, the authors proposed enriching MCTS with heuristic guidance and a database of decks to enhance performance. The approach was empirically validated against vanilla MCTS and state-of-the-art AI, showing significant performance gains. The integration of expert knowledge through heuristics and a deck database addressed two major challenges in Hearthstone: hidden information and a large search space. By incorporating a heuristic function into the selection and simulation steps, the algorithm could more effectively navigate the game's complex state space, leading to improved decision-making and performance. The use of a deck database allowed the MCTS algorithm to predict the opponent's cards more accurately, enhancing the quality of simulations and overall strategy. The empirical results demonstrated that the enhanced MCTS approach significantly outperformed vanilla MCTS and was competitive with existing AI players, showcasing its potential for complex, strategic games like Hearthstone. However, the reliance on pre-constructed heuristics and deck databases introduced additional complexity and potential limitations. The effectiveness of the heuristic function was highly dependent on its design and tuning, which might vary across different game scenarios and decks. Similarly, the deck database's accuracy and comprehensiveness were crucial for predicting opponent strategies, which might be challenging to maintain as new cards and strategies emerged. The additional computational overhead associated with managing and updating these enhancements could impact real-time performance, particularly in time-constrained gameplay environments. While the approach showed strong performance in controlled experiments, further validation in diverse and dynamic real-world scenarios was necessary to fully assess its robustness and adaptability.

Information Set MCTS (ISMCTS), an extension of MCTS designed for games with hidden information and uncertainty, was introduced in [170]. ISMCTS algorithms searched

trees of information sets rather than game states, providing a more accurate representation of games with imperfect information. The approach was tested across three domains with different characteristics. The ISMCTS algorithms excelled in efficiently managing hidden information by constructing trees where nodes represented information sets instead of individual states. This method mitigated the strategy fusion problem seen in determinization approaches, where different states in an information set were treated independently. By unifying statistics about moves within a single tree, ISMCTS made better use of the computational budget, leading to more informed decision-making. The empirical results demonstrated that ISMCTS outperformed traditional determinization-based methods in games with significant hidden information and partially observable moves. For instance, in Lord of the Rings: The Confrontation and the Phantom, ISMCTS achieved superior performance by effectively leveraging the structure of the game and reducing the impact of strategy fusion. However, the ISMCTS approach introduced additional complexity in maintaining and updating information sets, which could lead to increased computational overhead. The scalability of the method in environments with extensive hidden information and large state spaces remained a challenge, as the branching factor in information set trees could become substantial. While ISMCTS showed promising results in the tested domains, further validation in more diverse and dynamic scenarios was necessary to fully assess its robustness and general applicability. The reliance on accurate modeling of information sets and the necessity for domain-specific adaptations could limit the ease of implementation and the algorithm's flexibility across different types of games.

Authors in [171] investigated the application of MCTS to Kriegspiel, a variant of chess characterized by hidden information and dynamic uncertainty. The authors explored three MCTS-based methods, incrementally refining the approach to handle the complexities of Kriegspiel. They compared these methods to a strong minimax-based Kriegspiel program, demonstrating the effectiveness of MCTS in this challenging environment. The authors' incremental refinement of MCTS methods for Kriegspiel effectively addressed the game's dynamic uncertainty and large state space. By leveraging a probabilistic model of the opponent's pieces and incorporating domain-specific heuristics, the refined MCTS algorithms significantly improved performance compared to the initial naive implementation. The experimental results showed that the final MCTS approach outperformed the minimax-based program, achieving better strategic planning and decision-making. The innovative use of a three-tiered game tree representation and opponent modeling techniques demonstrated the adaptability and robustness of MCTS in handling partial information games. This study provided valuable insights into the application of MCTS in environments with incomplete information, highlighting its potential for broader applications in similar domains. Having said that, the reliance on extensive probabilistic modeling and heuristic adjustments introduced additional complexity, which could be computationally intensive and challenging

to maintain. The performance improvements were heavily dependent on the accuracy and relevance of the probabilistic models, which might vary across different game scenarios and opponents. While the approach showed strong performance in the controlled environment of Kriegspiel, its scalability and robustness in more diverse and dynamic real-world scenarios remained to be fully validated. The necessity for domain-specific knowledge to construct effective heuristics and models limited the generalizability of the method to other games with different characteristics. Further research was needed to explore the method's applicability and efficiency in a wider range of partial information games and real-world decision-making tasks.

Two extensions of MCTS tailored for Asymmetric Trees (MCTS-T) and environments with loops were introduced in [172]. The first algorithm, MCTS-T, addressed the challenges posed by asymmetric tree structures by incorporating tree uncertainty into the UCB formula, thus improving exploration. The second algorithm, MCTS-T+, further extended this approach to handle loops by detecting and appropriately managing repeated states within a single trace. The effectiveness of these algorithms was demonstrated through benchmarks on OpenAI Gym and Atari 2600 games. The key strength of MCTS-T lies in its ability to efficiently manage asymmetric trees by backing up the uncertainty related to the tree structure. This approach ensured that the algorithm could prioritize exploration in less explored subtrees, significantly improving the learning efficiency in environments with uneven tree depths. MCTS-T+ further enhanced this capability by addressing loops, preventing redundant expansions of the same state, and thus saving computational resources. The empirical results on benchmark tasks demonstrated that both MCTS-T and MCTS-T+ consistently outperformed standard MCTS, particularly in environments with strong asymmetry and loops. These improvements highlighted the potential of the proposed methods to extend the applicability of MCTS to more complex domains, such as robotic control and single-player video games, where asymmetry and looping were common. However, the reliance on non-stochastic environments and full state observability limited the generalizability of MCTS-T and MCTS-T+. These assumptions might not hold in many real-world scenarios, such as partially observable or highly stochastic environments, reducing the algorithms' applicability. The computational overhead associated with maintaining and updating the tree uncertainty and detecting loops could be substantial, particularly in large and dynamic state spaces. While the proposed methods showed promising results in controlled experiments, further validation in diverse real-world applications was necessary to fully assess their scalability and robustness. The complexity of tuning additional parameters, such as the uncertainty thresholds and loop detection mechanisms, posed additional challenges for practical implementation.

In the following subsection, we turn our attention to prioritized sweeping, another

algorithm within the category of model-based planning.

Prioritized Sweeping

Prioritized sweeping, introduced by [173], improves model-based RL by concentrating updates on the most important state-action pairs. This prioritization enhances both the speed of learning and convergence. As described in Alg. 19, the core mechanisms of prioritized sweeping include model learning, priority queue management, and backward sweeping. In model learning, the algorithm builds a model of the environment, capturing state transition probabilities and rewards (lines 3-6). A priority queue is used, where state-action pairs are prioritized based on the magnitude of their expected updates (error) (lines 7-8). Backward sweeping ensures that significant changes are promptly addressed by propagating their effects backward to earlier states (lines 9-17). The process starts with experience collection (lines 3-5), where the agent interacts with the environment to gather transitions (state, action, reward, next state). The model is then updated with these transitions (line 6). For each affected state-action pair, the algorithm calculates its priority (lines 7-8), reflecting the expected magnitude of the value update. The priorities are managed in a queue, and the algorithm updates the values in order of priority until the queue is empty or a convergence criterion is met (lines 9-17) [173, 174, 3].

Prioritized sweeping refines the Bellman equation by updating states in a priority-driven manner as follows:

$$P(s,a) = \left| R(s,a) + \gamma \left(\max_{a'} Q(s',a') \right) - Q(s,a) \right| \tag{3.6}$$

where $P(s,a)$ is the priority of the state-action pair (s,a). By reviewing several studies in the following paragraphs, the key advantage of prioritized sweeping becomes clear: its efficiency. Concentrating updates on high-priority areas minimizes unnecessary computations and accelerates the learning process. This prioritization also results in faster convergence compared to traditional DP methods. However, the algorithm introduces additional complexity by maintaining a priority queue and performing backward sweeps, which adds computational overhead. Moreover, the efficiency improvements are dependent on the accuracy of the model, which may not always be perfect in every scenario [173, 3].

Algorithm 19 Prioritized Sweeping [3]

```
Initialize Q(s, a), Model(s, a), for all s, a, and PQueue to empty
while true do                                                    ▷ Do forever
    S ← current (non-terminal) state
    A ← policy(S, Q)
    Execute action A; observe resultant reward, R, and state, S′
    Model(S, A) ← (R, S′)
    P ← |R + γ max_a Q(S′, a) − Q(S, A)|
    if P > θ then
        Insert (S, A) into PQueue with priority P
    end if
    for i = 1 to n do
        if PQueue is not empty then
            S, A ← first(PQueue)
            R, S′ ← Model(S, A)
            Q(S, A) ← Q(S, A)
            + α[R + γ max_a Q(S′, a) − Q(S, A)]
            for each (S̃, Ã) predicted to lead to S do
                R̃ ← predicted reward for (S̃, Ã, S)
                P ← |R̃ + γ max_a Q(S, a) − Q(S̃, Ã)|
                if P > θ then
                    Insert (S̃, Ã) into PQueue with
                    priority P
                end if
            end for
        end if
    end for
end while
```

The prioritized sweeping algorithm was introduced in [174] to improve learning efficiency in stochastic Markov systems. The method combined the rapid performance of incremental learning methods like TD and Q-learning with the accuracy of DP by prioritizing updates. By maintaining a priority queue, the algorithm processed the most impactful updates first, leading to faster convergence in large state-space environments. Although the algorithm showed superior performance in simulations, it involved significant computational overhead due to the priority queue maintenance and state tracking. The complexity of setting up and tuning the priority mechanism posed challenges, particularly in dynamic

environments with unpredictable state transitions.

An enhancement to the Deep Deterministic Policy Gradient (DDPG) (discussed in subsection 3.4.3) algorithm by incorporating prioritized sweeping for morphing UAVs is introduced in [175]. The objective is to improve the efficiency and effectiveness of morphing strategy decisions by avoiding the random selection of state-action pairs and instead focusing on those with a significant impact on learning outcomes. The integration of prioritized sweeping with the DDPG framework notably enhances learning efficiency by prioritizing state-action pairs that are most influential in updating the policy. This targeted approach accelerates convergence and improves decision-making accuracy, which is crucial for the dynamic and complex task of UAV morphing. The method effectively combines the strengths of value-based and Policy Gradient-based RL, leveraging Deep Neural Networks (DNNs) for state evaluation and policy improvement. The simulation results demonstrate that the improved algorithm achieves faster learning and higher total rewards compared to DDPG, validating its superiority in handling complex morphing scenarios. Despite its advantages, the reliance on accurate prioritization mechanisms introduces additional complexity in maintaining and updating the priority queue, which could impact performance in highly dynamic environments where state changes are rapid and unpredictable. The assumption that changes in sweep angle do not significantly affect flight status within short time frames may oversimplify real-world conditions, potentially limiting the model's accuracy. While the simulation results are promising, further validation in real-world applications is necessary to fully assess the method's robustness and scalability. The initial phase of training and data generation still requires significant computational resources, which could be a bottleneck in practical deployments.

Authors in [176] introduced modifications to Dyna-learning and prioritized sweeping algorithms, incorporating an epoch-incremental approach using Breadth-first Search (BFS). The combination of incremental and epoch-based updates improved learning efficiency, leading to faster convergence in dynamic environments like grid worlds. The use of BFS after episodes provided a more comprehensive understanding of the state space. However, managing dual modes of policy updates and accurate BFS calculations introduced complexity, potentially increasing computational overhead. The method showed strong simulation results, but real-world validation was necessary to understand its scalability and practical implications.

Authors in [177] introduced an innovative extension to the prioritized sweeping algorithm by employing small backups instead of full backups. The primary aim is to enhance the computational efficiency of model-based RL by reducing the complexity of value updates, making it more suitable for environments with a large number of successor states. The most notable advantage of this approach is its ability to perform value updates using

only the current value of a single successor state, significantly reducing the computation time. By utilizing small backups, the algorithm allows for finer control over the planning process, leading to more efficient update strategies. The empirical results demonstrate that the small backup implementation of prioritized sweeping achieves substantial performance improvements over traditional methods, particularly in environments where full backups are computationally prohibitive. The theoretical foundation provided in the paper supports the robustness of the small backup approach, ensuring that it maintains the accuracy of value updates while enhancing computational efficiency. Additionally, the parameter-free nature of small backups eliminates the need for tuning step-size parameters, which is a common challenge in traditional sample-based methods. On the other hand, the approach's reliance on maintaining and updating a priority queue introduces additional memory and computational overhead, particularly in environments with many state-action pairs. While the method improves computational efficiency, it requires careful management of memory resources to store component values associated with small backups. The simulations primarily focus on deterministic environments, leaving the performance of the small backup approach in highly stochastic or dynamic settings less explored. Further validation in real-world applications is necessary to fully assess the scalability and practicality of the method. Additionally, the initial phase of model construction and priority queue management could introduce latency, impacting the algorithm's real-time performance in complex environments.

Authors in [178] introduced cooperative prioritized sweeping to efficiently handle multi-agent MDPs using factored Q-functions and Dynamic Decision Networks (DDNs). CPS managed large state-action spaces effectively, leading to faster convergence in multi-agent environments. However, the reliance on accurately specified DDN structures and the batch update mechanism introduced complexity, increasing computational overhead in dynamic environments. While simulations showed cooperative prioritized sweeping's potential, further real-world validation was needed to assess its scalability and robustness.

A prioritized sweeping approach combined with Confidence-based Dual RL (CB-DuRL) for routing in Mobile Ad Hoc Networks (MANETs) is investigated by researchers in [179]. The proposed method dynamically selects routes based on real-time traffic conditions, aiming to minimize delivery time and congestion. The key strength of this approach is its dynamic adaptability to real-time traffic conditions, addressing the limitations of traditional shortest-path routing methods. By leveraging prioritized sweeping, the algorithm prioritizes updates to the most critical state-action pairs, enhancing learning efficiency and ensuring optimal path selection under varying network loads. The inclusion of CB-DuRL refines routing decisions by considering the reliability of Q-values, thus improving the robustness and reliability of the routing protocol. Empirical results from simulations

on a 50-node MANET demonstrate that the proposed method significantly outperforms traditional routing protocols in terms of packet delivery ratio, dropping ratio, and delay, showcasing its effectiveness in handling high traffic conditions and reducing network congestion. However, the approach's dependence on accurate estimation of traffic conditions and Q-values may limit its adaptability in highly dynamic or unpredictable environments where traffic patterns change rapidly. The initial phase of gathering sufficient data to populate the Q-tables and confidence values can introduce latency and computational overhead, potentially impacting the algorithm's performance in real-time applications. While the simulations provide strong evidence of the method's efficacy, further validation in larger and more diverse network topologies is necessary to fully assess its scalability and robustness. Managing priority queues and updating Q-values in a distributed manner across multiple nodes can also pose challenges in maintaining synchronization and consistency in real-world deployments.

A structured version of prioritized sweeping to enhance RL efficiency in large state spaces by leveraging Dynamic Bayesian Networks (DBNs) was introduced in [180]. The method accelerated learning by grouping states with similar values, reducing updates needed for convergence. DBNs provided a compact environment representation, further improving computational efficiency. However, the reliance on predefined DBN structures limited applicability in dynamic environments, and maintaining these structures added computational overhead. While promising in simulations, the method required further validation in diverse real-world scenarios to fully assess its scalability and robustness.

Dyna-Q

There are two variations of Dyna-Q that are commonly explored in RL: one using a tabular representation and the other utilizing function approximation. In this section, we analyze both variations of the Dyna-Q algorithm.

The Dyna-Q algorithm, introduced in [25], integrates traditional RL with planning, offering a novel approach to enhance the efficiency and adaptability of learning agents. This section provides an overview of the core concepts and mechanisms that underlie Dyna-Q. The algorithm combines learning from actual experiences with planning, utilizing a learned model of the environment. This integration enables the agent to improve its policy more quickly and efficiently by drawing from both real and simulated experiences. Dyna-Q leverages Q-learning [120], which combines immediate rewards with the discounted value of the next state, guiding the agent to take actions that maximize long-term rewards. Additionally, the Dyna-Q architecture includes a model of the environment that predicts the next state and reward given the current state and action. This model allows the agent

to simulate hypothetical experiences, enabling it to plan future actions without having to interact continuously with the real environment. The agent then uses the learned model to generate simulated experiences, treating them like real experiences to update the $Q(s, a)$ values and refine the agent's policy. This process accelerates learning by allowing the agent to practice and plan multiple scenarios internally.

The Dyna-Q planning process is incremental, meaning it can be paused and resumed at any time. This feature makes the algorithm highly adaptable to dynamic environments where the agent must continuously learn and adjust its policy based on new information. The algorithm operates through a series of steps, as shown in Alg. 20. First, the agent interacts with the environment, observes the current state, selects an action, receives a reward, and transitions to the next state (lines 3-5). Based on this experience, the $Q(s, a)$ value is updated (line 6). The environment model is then updated using the observed transition (s, a, r, s') (line 7), enhancing its predictions for future planning. The agent then uses this model to generate hypothetical experiences, selecting states and actions and predicting their outcomes (lines 8-14). These simulated experiences are used to further update the $Q(s, a)$ values, enabling the agent to refine its policy without additional real-world interactions.

Algorithm 20 Dyna-Q [25]

1: Initialize $Q(s, a)$ and Model(s, a) for all $s \in S$ and $a \in A(s)$
2: **repeat**
3: $S \leftarrow$ current (non-terminal) state
4: $A \leftarrow \epsilon$-greedy(S, Q)
5: Execute action A; observe resultant reward, R, and state, S'
6: $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
7: Model$(S, A) \leftarrow R, S'$ (assuming deterministic environment)
8: **for** $i = 1$ to n **do**
9: $S \leftarrow$ random previously observed state
10: $A \leftarrow$ random action previously taken in S
11: $R, S' \leftarrow$ Model(S, A)
12: $Q(S, A) \leftarrow Q(S, A) +$
$\alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
13: **end for**
14: **until** convergence or a stopping criterion is met

In the following paragraphs, we analyze several studies that have applied the Dyna-Q algorithm.

This paper [181] explored the development and application of the Dyna-H algorithm, a strategy that incorporated heuristic planning for optimizing decision-making in Role-playing Games (RPGs). The Dyna-H algorithm aimed to enhance path-finding efficiency by blending heuristic search with the model-free characteristics of RL, specifically within the context of the Dyna architecture. The primary advantage of the Dyna-H algorithm lies in its innovative combination of heuristic search and effectively addressing the challenges posed by large and complex search spaces typical in RPG scenarios. This synergy allowed the algorithm to focus on promising solution branches, significantly improving learning speed and policy quality compared to traditional methods like Q-learning and Dyna-Q. Additionally, the use of heuristic planning enabled the Dyna-H algorithm to perform efficiently without requiring a complete model of the environment beforehand, which was particularly beneficial for real-time applications. However, one drawback of the approach was the reliance on predefined heuristic functions, which might have limited its adaptability to various game environments where such heuristics were not readily available or accurate. The paper's evaluation, while thorough, focused primarily on deterministic grid-like environments, potentially overlooking the complexities and stochastic nature of more dynamic and unpredictable game settings. This narrow evaluation scope might have raised questions about the generalizability of the algorithm's performance in broader contexts.

In [182], a multi-objective Dyna-Q based Routing (MODQR) algorithm designed to optimize both delay and energy efficiency in wireless mesh networks was introduced. By leveraging the Dyna-Q technique, the algorithm aimed to address dynamic network conditions, ensuring that selected paths exhibited minimal delay and optimal energy usage. The key strength of the MODQR approach was its innovative integration of Dyna-Q with multi-objective optimization, effectively balancing delay and energy efficiency in real time. The use of Dyna-Q enhanced the convergence speed, making the algorithm well-suited for dynamic and complex network environments where conditions frequently change. Additionally, the consideration of both forward and reverse link conditions in determining path quality provided a more accurate assessment of route reliability. The inclusion of dynamic learning and exploration rates further refined the algorithm's adaptability, ensuring continuous improvement and optimization as the network evolved. However, the approach's reliance on predefined model parameters and assumptions, such as the gray physical interference model, might have limited its flexibility across different network topologies and conditions. While the Dyna-Q algorithm's fast convergence was advantageous, its performance in highly volatile environments with extreme fluctuations might not have been as robust. Moreover, the evaluation primarily focused on simulation results, which, while promising, might not have fully captured the complexities and unpredictable behaviors encountered in real-world deployments. The lack of comprehensive real-world testing left some uncertainty regarding the algorithm's practical effectiveness and scalability.

Authors in [183] presented a novel approach for managing Type-1 diabetes Mellitus through a Glycemic control system using Dyna-Q. The system aimed to automate insulin infusion based on Continuous Glucose Monitoring (CGM) data, eliminating the need for manual carbohydrate input from patients. The primary strength of this approach lies in its innovative use of the Dyna-Q algorithm, which combines model-based and model-free RL to enhance learning efficiency and control accuracy. By leveraging past CGM and insulin data, the system could predict future glucose levels and optimize insulin dosage in real time, resulting in improved Glycemic control. The algorithm's ability to operate effectively without explicit carbohydrate information significantly reduced the cognitive load on patients, aligning well with the goals of developing a fully automated artificial pancreas. Furthermore, the incorporation of a precision medicine approach tailored the model to individual patients, enhancing the adaptability and accuracy of the Glycemic predictions. On the other hand, the reliance on precise CGM data and insulin records for model training might have posed challenges in real-world scenarios where data quality could vary. While the algorithm showed promise in simulations and preliminary tests with real patients, its generalizability across diverse patient populations and long-term robustness remained areas of concern. The study's limited real-world testing meant that further extensive clinical trials were necessary to fully validate the system's effectiveness and safety in everyday use.

Dyna-T, which integrated Dyna-Q with Upper Confidence Bounds applied to Trees (UCT) for enhanced planning efficiency, was developed in [184]. The method aimed to improve action selection by using UCT to explore simulated experiences generated by the Dyna-Q model, demonstrating its effectiveness in several OpenAI Gym environments. The primary strength of Dyna-T was its ability to combine the model-based learning of Dyna-Q with the robust exploration capabilities of UCT. This combination allowed for a more directed exploration strategy, which enhanced the agent's ability to find optimal actions in complex and stochastic environments. The algorithm's use of UCT ensured that the most promising action paths were prioritized, significantly improving learning efficiency and convergence speed. The empirical results indicated that Dyna-T outperformed traditional RL methods, especially in environments with high variability and sparse rewards, showcasing its potential for broader applications in RL tasks. Despite its advantages, Dyna-T exhibited some limitations, particularly in deterministic environments where the added computational complexity of UCT might not have yielded significant benefits. The initial overhead of constructing and maintaining the UCT structure could be costly, especially in simpler tasks where traditional Dyna-Q or Q-learning might have sufficed. Moreover, while Dyna-T showed promise in simulated environments, its performance in real-world scenarios with continuous state and action spaces remained to be fully validated. The reliance on the exploration parameter 'c' and its impact on performance also warranted

further investigation to ensure robust adaptability across different problem domains.

A novel multi-path load-balancing routing algorithm for Wireless Sensor Networks (WSNs) leveraging the Dyna-Q algorithm was proposed in [185]. The proposed method, termed ELMRRL (Energy-efficient Load-balancing multi-path Routing with RL), aimed to minimize energy consumption and enhance network lifetime by selecting optimal routing paths based on residual energy, hop count, and energy consumption of nodes. The main advantage of the ELMRRL algorithm was its effective integration of Dyna-Q RL, which combined real-time learning with planning to adaptively select optimal routing paths. This dynamic adjustment ensured that the algorithm could respond to changes in network conditions, thus extending the network lifetime. The use of RL enabled each sensor node to act as an agent, making independent decisions based on local information, which was crucial for distributed environments like WSNs. Furthermore, the algorithm's focus on both immediate and long-term cumulative rewards led to more balanced energy consumption across the network, preventing premature node failures and improving overall network resilience. On the other hand, the reliance on local information for decision-making might have limited the algorithm's effectiveness in scenarios where global network knowledge could provide more optimal solutions. The initial learning phase, where nodes gathered and processed data to update their Q-values, could have introduced latency and computational overhead, potentially affecting the algorithm's performance in highly dynamic environments. Additionally, the approach's dependence on the correct parameterization of the reward function and learning rates could have been challenging, as these parameters significantly impacted the algorithm's efficiency and convergence speed. The paper primarily validated the algorithm through simulations, so its real-world applicability in diverse WSN deployments remained to be fully explored.

Researchers in [186] introduced a new motion control method for path planning in unfamiliar environments using the Dyna-Q algorithm. The goal of the proposed method was to improve the efficiency of motion control by combining direct RL with model learning, enabling agents to effectively navigate both dynamic and static obstacles. The main benefit of this method was its use of the Dyna-Q algorithm, which merged model-based and model-free RL techniques, facilitating concurrent learning and planning. This integration notably enhanced convergence speed and adaptability to dynamic environments, as demonstrated by faster path optimization compared to traditional Q-learning. The algorithm's capacity to simulate experiences allowed the agent to plan more efficiently, resulting in improved decision-making in complex situations. Furthermore, employing an ϵ-greedy policy for action selection ensured a balanced exploration-exploitation trade-off, which was essential for finding optimal paths in unknown environments. Nonetheless, the method's dependence on accurate environment models for effective planning might have restricted

its performance in highly unpredictable settings where models might not accurately reflect real-world dynamics. The computational burden of maintaining and updating the Q-table and environment model could also have been significant, especially in large state-action spaces. Additionally, while the method demonstrated promising results in simulation environments, its scalability and robustness in real-world applications with continuous state spaces and real-time constraints needed further validation. The reliance on predefined parameters, such as learning rates and discount factors, could also have impacted the algorithm's efficiency and effectiveness, requiring careful tuning for different scenarios.

Researchers in [187] introduced an improved Dyna-Q algorithm for mobile robot path planning in unknown, dynamic environments. The proposed method integrated heuristic search strategies, Simulated Annealing (SA), and a new action-selection strategy to enhance learning efficiency and path optimization. The enhanced Dyna-Q algorithm effectively merged heuristic search and SA, significantly improving the exploration-exploitation balance. By incorporating heuristic rewards and actions, the algorithm ensured efficient navigation through complex environments, avoiding local minima and achieving faster convergence. The novel SA-ϵ-greedy policy dynamically adjusted exploration rates, optimizing the learning process. Empirical results from simulations and practical experiments showed that the improved algorithm outperformed Q-learning and Dyna-Q methods, demonstrating superior global search capabilities, enhanced learning efficiency, and robust convergence properties in both static and dynamic obstacle scenarios. Despite the performance improvements from integrating heuristic search and SA, the approach's reliance on predefined heuristic functions might have limited its adaptability to diverse environments. The initial training phase required extensive exploration, potentially leading to higher computational overhead and longer training times. Additionally, the dependence on grid-based environment representation might have restricted the algorithm's scalability to continuous state spaces. The focus on simulated and controlled real-world environments raised concerns about the algorithm's robustness and generalizability in more complex and unpredictable real-world applications. Further studies were necessary to validate the method's effectiveness in larger, unstructured, and more dynamic environments.

An innovative approach for scheduling the charging of Plug-in Electric Vehicles (PEVs) using the Dyna-Q algorithm was developed in [188]. The primary goal was to minimize long-term charging costs while considering the stochastic nature of driving behavior, traffic conditions, energy usage, and fluctuating energy prices. The method formulated the problem as a MDP and employed DRL techniques for solution optimization. The key strength of this approach lies in its effective combination of model-based and model-free RL, which enhances learning speed and efficiency. By continuously updating the model with real experiences and generating synthetic experiences, the Dyna-Q algorithm ensured

rapid convergence to an optimal charging policy. This dual approach allowed for robust decision-making even in the face of uncertain and dynamic real-world conditions, such as varying energy prices and unpredictable driving patterns. Additionally, the integration of a deep-Q network for value approximation facilitated handling the vast state space inherent in PEV charging scenarios, ensuring the method's scalability and applicability to real-world settings. However, the reliance on accurate initial parameter values and predefined models for the user's driving behavior might have limited the algorithm's adaptability during the initial learning phase. The system's performance heavily depended on the quality and accuracy of these models, which might have varied across different users and environments. Moreover, while the approach demonstrated significant improvements in simulations, its effectiveness in diverse and more complex real-world scenarios required further validation. The potential computational overhead associated with maintaining and updating the deep-Q network and model could have posed challenges for real-time applications, especially in large-scale deployments with numerous PEVs.

Authors in [189] presented an enhanced Dyna-Q algorithm tailored for Automated Guided Vehicles (AGVs) navigating complex, dynamic environments. The key improvements included a global path guidance mechanism based on heuristic graphs and a dynamic reward function designed to address issues of sparse rewards and slow convergence in large state spaces. The improved Dyna-Q algorithm stood out by integrating heuristic graphs that provided a global perspective on path planning, significantly reducing the search space and enhancing efficiency. This method enabled AGVs to quickly orient towards their goals by leveraging precomputed shortest path information, thus mitigating the problem of sparse rewards commonly encountered in extensive environments. Additionally, the dynamic reward function intensified feedback, guiding AGVs more effectively through complex terrains and around obstacles. The experimental results in various scenarios with static and dynamic obstacles demonstrated superior convergence speed and learning efficiency compared to traditional Q-learning and standard Dyna-Q algorithms, highlighting its robustness and effectiveness in dynamic settings. However, the dependency on heuristic graphs, which required prior computation, might have limited the algorithm's adaptability in environments where real-time updates were necessary or in scenarios with unpredictable changes. The initial setup phase, involving the creation of the heuristic graph, could have introduced overheads that might not have been feasible for all applications. Furthermore, while the dynamic reward function enhanced learning efficiency, its design relied heavily on accurate modeling of the environment, which could have been challenging in highly variable or noisy conditions. The paper's focus on simulated environments left room for further validation in real-world applications, where additional factors such as sensor noise and real-time constraints could have impacted performance.

A Dyna-Q based anti-jamming algorithm designed to enhance the efficiency of path selection in wireless communication networks subject to malicious jamming was introduced in [190]. By leveraging both model-based and model-free RL techniques, the algorithm aimed to optimize multi-hop path selection, reducing packet loss and improving transmission reliability in hostile environments. The application of the Dyna-Q algorithm in this context was innovative, combining direct learning with simulated experiences to accelerate the convergence of the Q-values. This dual approach allowed the system to adapt quickly to dynamic jamming conditions, ensuring more reliable path selection and communication efficiency. The inclusion of a reward function that considered various modulation modes based on the signal-to-jamming noise ratio enhanced the robustness of the algorithm. Simulation results demonstrated that the Dyna-Q algorithm significantly outperformed traditional Q-learning and multi-armed bandit models, achieving faster convergence to optimal paths and better handling of interference, thus showcasing its potential for real-time applications in complex electromagnetic environments. Nevertheless, the method's reliance on pre-established environmental models might have limited its effectiveness in highly unpredictable or rapidly changing conditions, where the initial models might not accurately capture real-time dynamics. The need for accurate initial state representations and model updates introduced additional computational overhead, which could have impacted performance in larger or more complex networks. Furthermore, while the algorithm showed promise in simulated environments, its scalability and adaptability in real-world applications with varying node densities and jamming strategies required further validation. The focus on hop count minimization might also have overlooked other critical factors such as energy consumption and latency, which were essential for comprehensive network performance assessment.

Researchers in [191] introduced a model-based RL method using Dyna-Q tailored for multi-agent systems. It emphasized efficient environmental modeling through a tree structure and proposed methods for model sharing among agents to enhance learning speed and reduce computational costs. The approach leveraged the concept of knowledge sharing, where agents with more experience assisted others by disseminating valuable information. The integration of a tree-based model for environmental learning within the Dyna-Q framework significantly enhanced the efficiency of model construction and memory usage. This method allowed agents to generate virtual experiences, thus accelerating the learning process. The innovative model sharing techniques proposed in the paper, such as grafting partial branches and resampling, enabled agents to build more accurate models collaboratively. By reducing redundant data transfer and focusing on useful experiences, these sharing methods improved sample efficiency and learning speed in complex environments. The simulation results demonstrated the effectiveness of these techniques in multi-agent cooperation scenarios, highlighting their potential to optimize learning in large, continuous state spaces. Despite its advantages, the reliance on accurate decision tree models

for sharing experiences might have limited the approach's flexibility in highly dynamic or heterogeneous environments. The effectiveness of the sharing methods depended on the quality and relevance of the shared models, which might have varied across different agents and scenarios. Additionally, the initial phase of building accurate tree models could have been computationally intensive, particularly in environments with high variability. While the proposed methods showed promising results in simulations, further validation in diverse real-world applications was needed to fully assess their scalability and robustness. The paper also assumed a certain level of homogeneity among agents, which might not always have been applicable in more varied multi-agent systems.

The application of the Dyna-Q algorithm for path planning and obstacle avoidance in Unmanned Ground Vehicles (UGVs) and UAVs within complex urban environments was explored in [192]. The study focused on utilizing a vector field-based approach for effective navigation and air-ground collaboration tasks. The integration of the Dyna-Q algorithm with a vector field method significantly enhanced the efficiency and accuracy of path planning in dynamic urban settings. The approach leveraged both real and simulated experiences to adaptively update the agent's policy, ensuring rapid convergence to optimal paths. By simplifying the urban environment into a grid world, the method allowed for precise waypoint calculation, facilitating smooth navigation and effective obstacle avoidance. The use of PID controllers for UAV and UGV coordination further improved the stability and responsiveness of the system, enabling robust air-ground collaboration. Simulation results demonstrated that the proposed method effectively handled dynamic obstacles and complex path scenarios, showcasing its potential for real-world applications in urban environments. On the other hand, the reliance on grid-based environment representation might have limited the algorithm's scalability and adaptability to continuous state spaces found in more diverse and unstructured urban areas. The initial phase of creating the vector field and the grid map could have introduced computational overheads, which might have impacted real-time performance. While the paper focused on simulated environments, further validation in real-world scenarios was necessary to assess the approach's robustness and effectiveness under varying conditions. Additionally, the method's dependency on accurate dynamic models for both UGV and UAV could have posed challenges, as any discrepancies between the model and the real environment might have affected the overall performance.

This paper [193] evaluated the performance of the Dyna-Q algorithm in robot navigation within partially known environments containing static obstacles. The study extended the Dyna-Q algorithm to multi-robot systems and conducted extensive simulations to assess its efficiency and effectiveness. The primary strength of this study lies in its thorough analysis of the Dyna-Q algorithm in both single and multi-agent contexts. By integrating planning and model-based learning, the Dyna-Q algorithm sped up the learning process,

enabling robots to navigate efficiently even with limited prior knowledge of the environment. The use of simulations with the robot motion toolbox allowed for a comprehensive evaluation of the algorithm's performance across various parameters, providing valuable insights into the optimal settings for different scenarios. Extending the Dyna-Q algorithm to multi-robot systems showcased its adaptability and potential for complex task coordination, where agents could share knowledge to enhance overall system performance. However, the paper's focus on static obstacles and deterministic environments might have limited the applicability of the findings to more dynamic and stochastic settings. The initial need for environment discretization and model construction introduced additional computational overhead, which could have been a bottleneck in real-time applications. While the simulations offered a robust analysis, the lack of real-world validation left some uncertainty about the algorithm's practical effectiveness in unpredictable and continuously changing environments. Furthermore, the performance degradation observed in multi-agent scenarios indicated that further refinement was needed to improve coordination and reduce inter-agent interference.

The main advantage of Dyna-Q is its ability to accelerate learning by combining both real and simulated experiences. This dual approach reduces the reliance on extensive real-world interactions, making it ideal for applications where such interactions are costly or limited. Additionally, Dyna-Q is naturally adaptable to changing environments. The combination of continuous learning and planning allows the agent to dynamically update its policy in response to new information or environmental changes [25, 3]. Table **??** summarizes the studies that have applied model-based planning algorithms.

In the upcoming section, we present a detailed introduction to another paradigm of RL, policy-based methods, and analyze several algorithms that belong to this category.

3.3 Policy-based Methods

Policy-based methods are another core approach in RL, focusing more on directly optimizing the policy used to select actions for an agent. Unlike value-based methods, which first estimate the value function implicit in the task and then derive an optimal policy, policy-based methods directly parameterize and optimize the policy. This approach offers significant advantages, especially in challenging environments with high-dimensional action spaces or in cases where the policies are inherently stochastic. At the core of policy-based methods is the parameterization of policies, typically represented as $\pi(a|s;\theta)$, where θ denotes the policy parameters, s represents the state, and a is the action. In essence, the goal is to find the optimal parameters θ^* that maximize the expected cumulative reward. This

is typically achieved through gradient ascent techniques, specifically by Policy Gradient methods, which compute the gradient of the expected reward with respect to the policy parameters, adjusting them in the direction that increases the reward [3, 6, 194].

The gradient of the policy with respect to Q values is estimated in the following way:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) Q_{\pi_\theta}(s, a) \right] \tag{3.7}$$

Here, $Q_{\pi_\theta}(s, a)$ represents the action-value function under the current policy. This gradient can be estimated using samples from the environment, enabling the use of gradient ascent to iteratively update the policy parameters. The advantages of policy-based methods can be summarized as follows:

- **Direct Optimization of the Policy:** These methods optimize the policy directly, making them more effective than value-based methods, especially in environments with continuous and high-dimensional action spaces. They are particularly well-suited for problems in robotics and control, where actions are inherently continuous.
- **Stochastic Policies:** Policy-based methods naturally handle stochastic policies, which are useful in scenarios where exploration is crucial or when the optimal policy is stochastic by nature. Stochastic policies help to balance the exploration-exploitation trade-off more efficiently.
- **Improved Convergence Properties:** Policy-based methods often exhibit smoother convergence compared to value-based methods, which can struggle with instabilities and divergence due to challenges like the Church condition for value function approximations.

Now, we discuss the first Policy Gradient method, REINFORCE, in the next subsection.

3.3.1 REINFORCE

The **REINFORCE** algorithm is a foundational contribution to RL, especially within the realm of policy gradient methods. This algorithm aims to optimize the expected cumulative reward by adjusting the policy parameters in the direction of the gradient of the expected reward. As shown in Alg. 21, the REINFORCE algorithm operates within the stochastic policy framework, where the policy, parameterized by θ, defines a probability distribution

Algorithm 21 REINFORCE [3]

1: **Input:** a differentiable policy parameterization $\pi(a|s, \theta)$
2: Initialize policy parameter $\theta \in \mathbb{R}^{d'}$
3: **repeat** ▷ Forever:
4: Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot; \theta)$
5: **for** each step of the episode $t = 0, \ldots, T-1$ **do**
6: $G \leftarrow$ return from step t
7: $\theta \leftarrow \theta + \alpha\gamma^t G \nabla_\theta \ln \pi(A_t|S_t, \theta)$
8: **end for**
9: **until** convergence or a stopping criterion is met

over actions based on the current state. The central idea behind the REINFORCE algorithm is to use the log-likelihood gradient estimator to update the policy parameters [195]. The gradient of the expected reward with respect to the policy parameters θ is expressed as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\nabla_\theta \log \pi_\theta(a|s) G_t \right], \tag{3.8}$$

where $\pi_\theta(a|s)$ is the probability of taking action a in the state s under policy π parameterized by θ, and G_t is the return (cumulative future reward) following time step t. This gradient estimation forms the basis for the parameter update rule in REINFORCE (line 7):

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s) G_t, \tag{3.9}$$

where α is the learning rate.

One of the key strengths of the REINFORCE algorithm is its simplicity and generality, which makes it applicable to a wide variety of problems. However, it also encounters challenges, such as high variance in the gradient estimates and slow convergence, especially in environments with sparse or delayed rewards. By examining the applications of REINFORCE, we can gain a deeper understanding of these advantages and limitations.

Authors in [196] investigated the global convergence rates of the REINFORCE algorithm in episodic RL settings. The authors aimed to close the gap between theoretical and practical implementations of policy gradient methods by providing new convergence results for the REINFORCE algorithm. The paper's strengths lay in its comprehensive theoretical analysis and practical relevance. The authors derived performance bounds for

the REINFORCE algorithm using a fixed mini-batch size, aligning more closely with practical implementations. They provided the first set of global convergence results for the REINFORCE algorithm, including sub-linear high probability regret bounds and almost sure global convergence of average regret. The focus on the widely-used REINFORCE gradient estimation procedure rather than state-action visitation measure-based estimators addressed a significant gap between theory and practice. The authors established that the REINFORCE algorithm was sample efficient, with polynomial complexity, which was crucial for practical applications due to the high cost of obtaining samples. However, the complexity of the analysis might have challenged practitioners less familiar with the mathematical concepts. The reliance on assumptions such as the log-barrier regularization term and soft-max policy parametrization might have limited the generality of the results. The paper focused on the stationary infinite-horizon discounted setting, and a more detailed discussion on applying the results to other settings would have enhanced its relevance. The absence of empirical validation of the proposed convergence bounds and sample efficiency was another limitation. Including experimental results would have provided additional evidence of the practical utility of the findings.

A novel approach to Energy Management Strategies (EMS) in Fuel Cell Hybrid EVs (FCHEV) using the fuzzy REINFORCE algorithm was introduced in [197]. This method integrated a fuzzy inference system (FIS) with Policy Gradient RL (PGRL) to optimize energy management, achieve hydrogen savings, and maintain battery operation. One of the key strengths of the paper was the innovative combination of fuzzy logic with the REINFORCE algorithm. By employing a fuzzy inference system to approximate the policy function, the authors effectively leveraged the generalization capabilities of fuzzy logic to handle the complexity and uncertainty inherent in energy management tasks. This integration helped to address the limitations of traditional EMS methods that relied heavily on expert knowledge and static rules, thus providing a more adaptive and robust solution. The use of a fuzzy baseline function to stabilize the training process and reduce the variance in policy gradient updates was another notable advantage. This approach enhanced the convergence rate and stability of the learning process, which was particularly beneficial in real-time applications where computational efficiency and robustness were critical. The paper's demonstration of the algorithm's adaptability to changing driving conditions and system states further underscored its practical relevance and effectiveness. However, the complexity of the proposed method might have posed implementation challenges, particularly for practitioners who were less familiar with fuzzy logic. The integration of FIS and PGRL required careful tuning of parameters and membership functions, which could have been time-consuming and computationally intensive. Additionally, while the fuzzy REINFORCE algorithm showed promise in reducing the computational burden and improving convergence, the reliance on fuzzy logic introduced an additional layer of complexity that

might not have been necessary for all applications. The paper also provided a comprehensive analysis of the simulation and hardware-in-loop experiments, validating the effectiveness of the proposed method in real-world scenarios. The results indicated that the fuzzy REINFORCE algorithm could achieve near-optimal performance without requiring accurate system models or extensive prior knowledge, making it a versatile and practical solution for EMS in FCHEVs.

Authors in [198] presented a study protocol for a trial aimed at improving medication adherence among patients with type 2 diabetes using an RL-based text messaging program. One of the key strengths of this study was its innovative use of RL to personalize text message interventions. By tailoring messages based on individual responses to previous messages, the approach had the potential to optimize engagement and improve adherence more effectively than generic messaging strategies. This personalized communication could have led to more significant behavior changes and better health outcomes for patients with diabetes. The study's design also enhanced its practical relevance. Conducted in a real-world setting at Brigham and Women's Hospital, it involved patients with suboptimal diabetes control, which reflected a common clinical scenario. The use of electronic pill bottles to monitor adherence provided accurate and objective data, supporting the reliability of the study outcomes. Additionally, the trial's primary outcome of average medication adherence over six months was a meaningful measure that directly related to the study's objective. However, there were some weaknesses and challenges associated with the study. The requirement for patients to use electronic pill bottles and smartphones with a data plan or WiFi might have limited the generalizability of the findings to populations without access to such technology. Furthermore, the study's reliance on self-reported adherence as a secondary outcome introduced the potential for reporting bias. The study also faced potential limitations related to the length of the follow-up period and the evaluation of the long-term sustainability of the intervention. While a six-month follow-up period was sufficient to assess initial adherence improvements, longer-term studies would have been necessary to determine whether the benefits of the intervention were sustained over time.

In [199], the authors presented a novel method for rate adaptation in 802.11 wireless networks leveraging the REINFORCE algorithm. The proposed approach, named ReinRate, integrated a comprehensive set of observations, including received signal strength, contention window size, current modulation and coding scheme, and throughput, to adapt dynamically to varying network conditions and optimize network throughput. One of the key strengths of this paper was its innovative application of the REINFORCE algorithm to WiFi rate adaptation. Traditional rate adaptation algorithms like Minstrel and Ideal relied on limited observations such as packet loss rate or signal-to-noise ratio, which could have been insufficient in dynamic wireless environments. In contrast, ReinRate's broader set

of observations allowed for a more nuanced response to varying conditions, leading to significant improvements in network performance. The authors demonstrated that ReinRate outperformed Minstrel and Ideal algorithms by up to 102.5% and 30.6% in network scenarios without interference, and by up to 35.1% and 66.6% in scenarios with interference. Another strength was the comprehensive evaluation of ReinRate using the ns-3 network simulator and ns3-ai OpenAI Gym. The authors conducted extensive simulations under various network scenarios, both static and dynamic, with and without interference. This thorough evaluation provided strong evidence of the algorithm's effectiveness and adaptability in real-world conditions. The results indicated that ReinRate consistently achieved higher throughput compared to traditional algorithms, showcasing its ability to handle the challenges of dynamically changing wireless environments. However, the complexity of the proposed method might have posed challenges for practical implementation. The integration of multiple observations and the application of the REINFORCE algorithm required careful tuning of parameters and computational resources.

A new approach to enhance DRL for outdoor robot navigation was investigated in [200]. The key innovation was the use of a heavy-tailed policy parameterization, which induced exploration in sparse reward settings, a common challenge in outdoor navigation tasks. A significant strength of the paper lies in addressing the sparse reward issue, which was prevalent in many real-world navigation scenarios. Traditional DRL methods often relied on carefully designed dense reward functions, which could have been impractical to implement. The authors proposed HTRON, an algorithm that leveraged heavy-tailed policy parameterizations, such as the Cauchy distribution, to enhance exploration without needing complex reward shaping. This approach allowed the algorithm to learn efficient behaviors even with sparse rewards, making it more applicable to real-world scenarios. The paper's thorough experimental evaluation was another strong point. The authors tested HTRON against established algorithms like REINFORCE, Proximal Policy Optimization (PPO), and Trust Region Policy Optimization (TRPO) (explained later in the upcoming subsections) across three different outdoor scenarios: goal-reaching, obstacle avoidance, and uneven terrain navigation. HTRON outperformed these algorithms in terms of success rate, average time steps to reach the goal, and elevation cost, demonstrating its effectiveness and efficiency. The use of a realistic unity-based simulator and the deployment of the algorithm on a Clearpath Husky robot further validated the practical applicability of the proposed method. However, the complexity of the proposed algorithm and the specific choice of heavy-tailed distributions might have posed challenges. The implementation of heavy-tailed policy gradients could have introduced instability, especially in the initial learning phases. While the authors mitigated this with adaptive moment estimation and gradient clipping, these techniques required careful tuning and expertise, potentially limiting accessibility for practitioners.

The next policy-based algorithm we discuss is TRPO. In the following subsection, we will cover this algorithm.

3.3.2 Trust Region Policy Optimization (TRPO)

TRPO, introduced by [201], represents a significant advancement in RL, particularly within policy optimization methods. The main goal of TRPO is to optimize control policies while guaranteeing monotonic improvement, addressing the weaknesses of earlier methods [202, 203, 204] that often led to unstable policy updates and poor performance in complex tasks. TRPO is specifically designed to handle large, nonlinear policies, such as those represented by neural networks. The algorithm ensures that each policy update results in a performance improvement by keeping the updated policy within a "trust region" around the current policy. This trust region is defined through a constraint on the KL divergence between the new and old policies, effectively preventing large, destabilizing updates [205, 201]. It operates within the stochastic policy framework, where the policy π_θ is parameterized by θ and defines a probability distribution over actions given the states. The expected discounted reward for a policy π is given by:

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right], \tag{3.10}$$

where γ is the discount factor, $r(s_t, a_t)$ is the reward at time step t, and the expectation is taken over the state and action trajectories induced by the policy. To ensure that the policy update remains within a safe boundary, TRPO constrains the KL divergence between the new policy $\pi_{\theta'}$ and the old policy π_θ:

$$D_{KL}(\pi_\theta \| \pi_{\theta'}) \leq \zeta, \tag{3.11}$$

where ζ is a small positive constant. This constraint ensures that the new policy does not deviate too much from the old policy, thereby providing stability to the learning process. TRPO optimizes a surrogate objective function that approximates the true objective while respecting the trust region constraint. The surrogate objective $L(\theta)$ is defined as:

$$L(\theta) = \hat{\mathbb{E}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} \hat{A}_{\theta_{\text{old}}}(s, a)\right], \tag{3.12}$$

where $\hat{A}_{\theta_{\text{old}}}(s, a)$ is an estimate of the advantage function, which measures the relative value of taking action a in state s under the old policy. As demonstrated in Alg. 22, the practical implementation of TRPO involves the following steps:

Algorithm 22 TRPO [206]

1: **Input:** initial policy parameters θ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
4: Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
5: Form sample estimates for:
6: policy gradient $\hat{g}_k$ (using advantage estimates)
7: and KL-divergence Hessian-vector product function $f(v) = \hat{H}_k v$
8: Use CG with n_{cg} iterations to obtain $x_k \approx \hat{H}_k^{-1} \hat{g}_k$
9: Estimate proposed step $\Delta_k \approx \sqrt{\frac{2\delta}{x_k^T \hat{H}_k x_k}} x_k$
10: Perform backtracking line search with exponential decay to obtain final update
11: $\theta_{k+1} = \theta_k + \alpha^j \Delta_k$
12: **end for**

1. **Sample Trajectories:** Collect a set of trajectories using the current policy $\pi_{\theta_{\text{old}}}$ (line 3).

2. **Estimate Advantages:** Compute the advantage function $\hat{A}_{\theta_{\text{old}}}(s, a)$ using the collected trajectories (line 4).

3. **Optimize Surrogate Objective:** Solve the constrained optimization problem to find the new policy parameters θ' (lines 5-7):

$$\theta' = \arg\max_{\theta} \hat{\mathbb{E}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} \hat{A}_{\theta_{\text{old}}}(s, a) \right], \tag{3.13}$$

subject to

$$D_{KL}(\pi_{\theta_{\text{old}}} \| \pi_\theta) \leq \zeta. \tag{3.14}$$

4. **Update Policy:** Update the policy parameters to θ' (lines 9-11).

Let us examine several research studies that have utilized TRPO to gain a deeper understanding of its applications and effectiveness. A Monotonic Policy Optimization (MPO) algorithm was designed to address the challenges associated with high-dimensional continuous control tasks in [207]. The primary focus was on ensuring monotonic improvement in policy performance, which was crucial for stability and efficiency in RL. One of the significant strengths of this paper was its innovative approach to policy optimization. The authors derived a new lower bound on policy improvement that penalized average policy

divergence on the state space rather than the maximum divergence. This approach addressed a critical limitation of previous algorithms, such as TRPO, which could suffer from worst-case degradation in policy performance. By focusing on average divergence, the MPO algorithm ensured more consistent and reliable improvements, making it particularly suitable for high-dimensional continuous control tasks. The empirical evaluation of the MPO algorithm was another strong point. The authors conducted extensive simulations using the MuJoCo physics engine, testing the algorithm on various challenging robot locomotion tasks, including swimming, quadruped locomotion, bipedal locomotion, and more. The results demonstrated that the MPO algorithm consistently outperformed state-of-the-art methods like TRPO and Vanilla policy gradient in terms of average discounted rewards. The algorithm's performance in these high-dimensional tasks highlighted its robustness and practical applicability. However, the complexity of the proposed algorithm might have posed challenges for practical implementation. The need for computing the natural policy gradient and performing line searches to determine optimal step sizes required significant computational resources. This complexity could have limited the accessibility of the MPO algorithm for practitioners without advanced computational capabilities. Additionally, while the algorithm guaranteed monotonic improvement, this came at the cost of slower training speed due to conservative step sizes. The authors suggested combining MPO with other faster methods during the initial training phase to mitigate this issue, but this added another layer of complexity to the implementation.

[208] introduced an enhancement to the TRPO algorithm by incorporating entropy regularization. This modification, termed EnTRPO, aimed to improve exploration and generalization by encouraging more stochastic policy choices. The paper demonstrated the effectiveness of EnTRPO through experiments on the CartPole system, showcasing better performance compared to the original TRPO. One of the main strengths of the paper was its innovative use of entropy regularization to enhance the TRPO algorithm. By adding an entropy term to the advantage function, the authors effectively encouraged exploration, which was crucial to avoid premature convergence to suboptimal policies. This approach addressed a common limitation of TRPO, which could sometimes restrict exploration due to its strict KL divergence constraints between consecutive policies. The entropy regularization helped maintain a balance between exploration and exploitation, leading to more robust learning outcomes. The empirical evaluation provided in the paper was another significant strength. The authors conducted thorough experiments using the Cart-Pole system, a well-known benchmark in the field. The results showed that EnTRPO converged faster and more reliably than TRPO, particularly when the discount factor was set to 0.85. This indicated that the proposed method not only improved exploration but also enhanced the overall convergence speed and stability of the learning process. The use of a well-defined experimental setup, including details on neural network architectures and

hyperparameters, added credibility to the findings. A potential limitation was the reliance on a single benchmark task for evaluation. While the Cart-Pole system was a standard benchmark, it was relatively simple compared to many real-world applications. The paper would have benefited from additional experiments on more complex tasks and environments to demonstrate the generalizability and robustness of EnTRPO. This would have provided stronger evidence of the method's effectiveness across a wider range of scenarios.

The challenge of applying trust region methods to Multi-Agent RL (MARL) was investigated in [209]. The authors introduced Heterogeneous-agent TRPO (HATRPO) and Heterogeneous-Agent Proximal Policy Optimization (HAPPO) algorithms. These methods were designed to guarantee monotonic policy improvement without requiring agents to share parameters or relying on restrictive assumptions about the decomposability of the joint value function. A strength of the paper was its theoretical foundation. The authors extended the theory of trust region learning to cooperative MARL by developing a multi-agent advantage decomposition lemma and a sequential policy update scheme. This theoretical advancement allowed HATRPO and HAPPO to ensure monotonic improvement in joint policy performance, a key advantage over existing MARL algorithms that did not guarantee such improvement. This theoretical guarantee was essential for stable and reliable learning in multi-agent settings, where individual policy updates could often lead to non-stationary environments and suboptimal outcomes. The empirical validation of HATRPO and HAPPO on benchmarks such as multi-Agent MuJoCo and StarCraft II demonstrated the effectiveness of these algorithms. The results showed that HATRPO and HAPPO significantly outperformed strong baselines, including Independent Proximal Policy Optimization (IPPO), MAPPO, and MADDPG, in various tasks. This performance improvement highlighted the practical applicability of the proposed methods in complex, high-dimensional environments. The thorough experimental evaluation across multiple scenarios provided strong evidence of the robustness and generalizability of HATRPO and HAPPO. However, the complexity of implementing HATRPO and HAPPO could have been a potential limitation. The algorithms required the computation of multi-agent advantage functions and sequential updates, which could have been computationally intensive and challenging to implement efficiently. This complexity might have limited the accessibility of these methods to practitioners who might not have had advanced computational resources or expertise in implementing sophisticated algorithms.

Authors in [210] presented a method to actively recognize objects by choosing a sequence of actions for an active camera. This method utilized TRPO combined with Extreme Learning Machines (ELMs) to enhance the efficiency of the optimization algorithm. One of the significant strengths of this paper was its innovative application of TRPO in conjunction with ELMs. ELMs provide a simple yet effective way to approximate poli-

cies, reducing the computational complexity compared to traditional DNNs. This resulted in an efficient optimization process, crucial for real-time applications like active object recognition. The use of ELMs allowed for faster convergence and more straightforward implementation, making the proposed method accessible for practical applications. However, the complexity of integrating TRPO with ELMs could have posed challenges for some practitioners. Although ELMs simplified the optimization process, they still required careful tuning of parameters, such as the number of hidden nodes and the distribution of random weights. This additional layer of complexity might have limited the method's accessibility for users without extensive experience in RL and NNs.

In [211], the authors explored the application of the TRPO algorithm in MARL environments, specifically focusing on hide-and-seek games. The authors compared the performance of TRPO with the Vanilla Policy Gradient (VPG) algorithm to determine the most effective method for this type of game. One of the primary strengths of this paper was its focus on a well-defined, complex, multi-agent environment. Hide-and-seek games inherently involve dynamic interactions between agents, making them an excellent testbed for evaluating algorithms. By using TRPO, which was designed to ensure monotonic policy improvement, the authors addressed a significant challenge in MARL: maintaining stable and consistent learning despite the presence of multiple interacting agents. The empirical results presented in the paper highlighted the strengths of TRPO, especially in scenarios where the testing environment differed from the training environment. TRPO's ability to adapt to new environments and maintain high performance was a notable advantage over the VPG algorithm, which performed better in environments identical to the training conditions but struggled when faced with variability. This adaptability was crucial for practical applications of MARL, where agents often encountered unpredictable changes in their environment. Another strength was the comprehensive experimental setup, which included various configurations and scenarios. The authors meticulously compared the performance of TRPO and VPG across different numbers of agents and types of environments (quadrant and random walls scenarios). This thorough approach provided robust evidence supporting the efficacy of TRPO in MARL settings. However, the paper also had some limitations. The complexity of implementing TRPO in a multi-agent context could have been a barrier for practitioners. TRPO required careful tuning and substantial computational resources, which might not have been readily available in all settings. Additionally, the reliance on simulation results raised questions about the real-world applicability of the findings. While the hide-and-seek game was a useful simulation environment, real-world deployments could have presented additional challenges not captured in the simulations.

The application of TRPO to improve Cross-Site Scripting (XSS) detection systems was analyzed by authors in [212]. The authors aimed to enhance the resilience of XSS filters

against adversarial attacks by using RL techniques to identify and counter malicious inputs. One of the main strengths of this paper was its innovative approach to applying TRPO in Cybersecurity, specifically for XSS detection. Traditional XSS detection methods often relied on static rules and signatures, which could be easily bypassed by sophisticated attackers. By leveraging TRPO, the authors introduced a dynamic and adaptive mechanism that could learn to detect and counteract adversarial attempts to exploit XSS vulnerabilities. This use of TRPO enhanced the robustness of the detection system, making it more resilient to evolving threats. A limitation of this study was the reliance on specific hyperparameters, such as the learning rate and discount factor, which could have significantly impacted the model's performance. The paper would have benefited from a more detailed discussion on how these parameters were selected and their influence on the detection model. Providing guidelines or heuristics for parameter tuning would have helped practitioners replicate and extend the study's findings.

Authors in [213] aimed to create a universal policy for a locomotion task that could adapt to various robot morphologies using TRPO. The study investigated the use of surrogate models, specifically Polynomial Chaos Expansion (PCE) and model ensembles, to model the dynamics of the robots. One of the primary strengths of this thesis was its innovative approach to developing a universal policy. The use of TRPO ensured stability and reliable policy updates even in complex environments. The focus on creating a policy that could generalize across different robot configurations was particularly noteworthy, as it addressed the challenge of designing controllers that were not limited to a single robot morphology. The integration of surrogate models, especially the PCE, was another strong point. PCE allowed for efficient sampling and modeling of the stochastic environment, potentially reducing the number of interactions required with the real environment. This was crucial for practical applications where real-world interactions could have been costly or risky. The theoretical foundation laid for using PCE in this context was robust and showed promise for future research. However, the thesis also highlighted several challenges and limitations. The complexity of accurately modeling the dynamics with PCE was a significant hurdle. The results indicated that while PCE showed potential, it currently could not model the dynamics accurately enough to be used in combination with TRPO effectively. The computational time required for PCE was also a practical concern, limiting its immediate applicability. The model ensemble surrogate showed some promise but ultimately failed to train a successful policy. This pointed to the difficulty of creating surrogate models that could capture the complexities of robot dynamics sufficiently. The thesis suggested that using the original environment from the RoboGrammar library yielded better results, emphasizing the need for more advanced or alternative surrogate modeling techniques.

A novel approach for optimizing Home Energy Management Systems (HEMS) using Multi-agent TRPO (MA-TRPO) was investigated in [214]. This approach aimed to improve energy efficiency, cost savings, and consumer satisfaction by leveraging TRPO techniques in a multi-agent setup. One of the primary strengths of this paper was its consumer-centric approach. Traditional HEMS solutions often prioritized energy efficiency and cost savings without adequately considering consumer preferences and comfort. By incorporating a preference factor for Interruptible-Deferrable Appliances (IDAs), the proposed MA-TRPO algorithm ensured that consumer satisfaction was taken into account, leading to a more holistic and practical solution. This consumer-centric focus was crucial for the widespread adoption of HEMS in real-world settings. Another strength was the comprehensive use of real-world data for training and validation. The authors utilized five-minute retail electricity prices derived from wholesale market prices and real-world Photovoltaic (PV) generation profiles. This approach enhanced the practical relevance and robustness of the proposed method, as it demonstrated the algorithm's effectiveness under realistic conditions. Additionally, the paper provided a detailed explanation of the various components of the smart home environment, including the non-controllable base load, IDA, Battery Energy Storage System (BESS), and PV system, which added clarity and depth to the study. The use of MA-TRPO in a multi-agent setup was also a significant contribution. The proposed method modeled and trained separate agents for different components of the HEMS, such as the IDA and BESS, allowing for more specialized and effective control strategies. This multi-agent approach addressed the complexities and inter-dependencies within the home energy environment, leading to more efficient and coordinated energy management. The paper's reliance on simulation results, while comprehensive, still left questions about the real-world applicability of the proposed method. Although the use of real-world data enhanced relevance, further validation in actual home environments would have strengthened the case for practical deployment. Real-world testing was essential to ensure the robustness and effectiveness of the MA-TRPO algorithm in diverse and dynamic home energy scenarios. Additionally, the reliance on discrete action spaces simplified the problem but might not have fully captured the nuances of continuous control in real-world applications. Future work could explore extending the algorithm to handle continuous action spaces for more precise control.

Authors in [215] investigated the application of TRPO to address the joint spectrum and power allocation problem in the Internet of Vehicles (IoV). The objective was to minimize AoI and power consumption, which were crucial for maintaining real-time communication and energy efficiency in vehicular networks. One of the key strengths of this paper was its focus on AoI, a vital metric for ensuring timely and accurate information exchange in vehicular communications. By incorporating AoI into the optimization framework, the authors addressed a significant challenge in IoV networks, where the freshness of information

directly impacted road safety and traffic efficiency. The proposed TRPO-based approach effectively balanced the trade-off between minimizing AoI and reducing power consumption, showcasing its practical relevance. The paper's reliance on certain assumptions, such as the availability of CSI and the periodic reporting of CSI to the base station, might have limited its generalizability. In real-world scenarios, obtaining accurate and timely CSI could have been challenging due to various factors like signal interference and mobility. Future work could explore more practical approaches to CSI estimation and reporting to enhance the applicability of the proposed solution.

An advancement over TRPO was proposed as a new algorithm, PPO.

3.3.3 Proximal Policy Optimization (PPO)

PPO, introduced by [216], marks a significant advancement in policy gradient methods. PPO is designed to achieve reliable performance and sample efficiency, addressing the limitations of earlier policy optimization algorithms, such as VPG methods and TRPO. Using policy gradient methods, the policy parameters are optimized through stochastic gradient ascent by estimating the gradient of the policy. One of the most widely used policy gradient estimators is:

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}_t \right], \tag{3.15}$$

where π_θ represents the policy parameterized by θ, and $\hat{A}_t$ is an estimator of the advantage function at time step t. This estimator helps construct an objective function whose gradient corresponds to the policy gradient estimator:

$$L_{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t|s_t) \hat{A}_t \right]. \tag{3.16}$$

PPO simplifies TRPO by utilizing a surrogate objective with a clipped probability ratio, enabling multiple epochs of mini-batch updates. To ensure stable learning, large policy updates should be avoided. Therefore, the PPO objective is defined as follows:

$$L_{CLIP}(\theta) = \hat{\mathbb{E}}_t \Big[\min \big(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \big) \Big] \tag{3.17}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio, and ϵ is a hyperparameter. This objective clips the probability ratio to ensure it stays within a reasonable range, preventing excessively large updates.

Figure 3.4: Plots show L_{CLIP} for one timestep against the probability ratio r, with positive and negative advantages. The red circle marks the start ($r = 1$), and L_{CLIP} sums these terms [216]

The practical implementation of PPO involves the following steps:

1. **Sample Trajectories:** Collect trajectories using the current policy $\pi_{\theta_{\text{old}}}$.
2. **Estimate Advantages:** Compute the advantage function $\hat{A}_t$ using the collected trajectories.
3. **Optimize Surrogate Objective:** Perform several epochs of optimization on the surrogate objective $L_{CLIP}(\theta)$ using minibatch stochastic gradient descent.
4. **Update Policy:** Update the policy parameters to the new parameters θ.

Before diving into related papers, to illustrate the algorithm's functionality, PPO's algorithm is provided in Alg. 23. A DRL approach for optimizing traffic flow in mixed-autonomy scenarios, where both Connected Autonomous Vehicles (CAVs) and human-driven vehicles coexisted, was introduced in [217]. The authors proposed three distributed learning control policies for CAVs using PPO, a policy gradient DRL method, and conducted experiments with varying traffic settings and CAV penetration rates on the flow

Algorithm 23 PPO [216]

Initialize policy parameters θ, and value function parameters ϕ
Initialize old policy $\pi_{\theta_{\text{old}}} \leftarrow \pi_\theta$
for iteration $i = 1$ to M **do**
  Collect trajectories $\{(s_t, a_t, r_t, s_{t+1})\}$ by running policy $\pi_{\theta_{\text{old}}}$
  Compute advantage estimates $\hat{A}_t$ for each trajectory using the value function
  for epoch $j = 1$ to K **do**
    Compute probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$
    Define the surrogate loss:

$$L_{CLIP}(\theta) = \hat{\mathbb{E}}_t\Big[\min\Big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\Big)\Big]$$

    Perform mini-batch gradient ascent on the surrogate objective $L_{CLIP}(\theta)$
    Update the value function parameters ϕ by minimizing:

$$L_V(\phi) = \hat{\mathbb{E}}_t\left[(V_\phi(s_t) - R_t)^2\right]$$

  end for
  Update old policy: $\pi_{\theta_{\text{old}}} \leftarrow \pi_\theta$
end for

framework, a new open-source microscopic traffic simulator. One of the primary strengths of the paper was its innovative approach to mixed-autonomy traffic optimization at a network level. The authors hypothesized that controlling distributed CAVs at a network level could outperform individually controlled CAVs, and their experimental results supported this hypothesis. The network-level RL policies for controlling CAVs significantly improved the total rewards and average velocity compared to individual-level RL policies. This finding was crucial for the development of ITS, which aimed to optimize traffic flow and reduce congestion. The use of the flow framework and SUMO environment for experiments was another strength of the paper. These tools provided a realistic and flexible simulation environment for testing the proposed control policies. The comprehensive evaluation, including different traffic settings and CAV penetration rates (10%, 20%, and 30%), added robustness to the findings. The authors' use of three different learning strategies—single-agent asynchronous learning, joint global cooperative learning, and joint local cooperative

learning—allowed for a thorough comparison of the effectiveness of network-level control policies. The paper also highlighted the potential high communication overhead associated with the global joint cooperative policy, especially as the penetration rate of CAVs increased. This overhead could have limited the scalability of the solution in real-world scenarios where communication resources were constrained. The authors suggested that the local joint cooperative policy, which required less communication, could have been a more practical choice in such situations, though it might not have performed as well as the global policy.

In [218], the authors presented a novel method for managing power grid line flows in real time using PPO. This approach addressed the challenges posed by the increasing penetration of renewable resources and the unpredictability of power system conditions, aiming to prevent line overloading and potential cascading outages. One of the key strengths of this paper was the application of PPO in a critical real-time grid operation context. PPO was well-suited for this task due to its ability to ensure stable updates within trust regions, thus maintaining the reliability of the control policies. This characteristic was crucial in power systems, where stability and safety were paramount. The authors' decision to use PPO over other algorithms like DDPG and Policy Gradient was justified by PPO's robustness in handling larger and more complex action spaces. A limitation was the reliance on simulation results for validation. While the use of high-fidelity power grid simulators was a strength, real-world deployments could present additional challenges not captured in simulations. Factors such as communication delays, unexpected equipment failures, and human operator interventions could have impacted the effectiveness of the control strategies. Further real-world testing and validation were necessary to fully assess the robustness and practicality of the proposed method in diverse and dynamic power grid environments.

A framework for optimizing metro service schedules and train compositions using PPO within a DRL framework was proposed in [219]. This method was applied to handle the dynamic and complex problem of metro train operations, focusing on minimizing operational costs and improving service regularity. A significant strength of the paper was its innovative application of PPO to metro service scheduling and train composition. PPO, known for its stability and efficiency in handling large-scale optimization problems, was effectively utilized to address the dynamic nature of metro operations. The integration of PPO with ANNs for approximating value functions and policies demonstrated a robust approach to tackling the high-dimensional decision space inherent in metro scheduling. This combination enhanced the framework's ability to adapt to varying passenger demands and operational constraints. The paper's use of a real-world scenario, specifically the Victoria line of the London underground, for testing and validation was another strong point. The authors provided a comprehensive evaluation, comparing their PPO-based method

with established meta-heuristic algorithms like the genetic algorithm and differential evolution. The results indicated that the PPO-based approach outperformed these traditional methods in terms of both solution quality and computational efficiency. This practical validation underscored the method's applicability and effectiveness in real-world settings. The reliance on a specific set of operational constraints, such as fixed headways and trailer limits, might have also limited the method's generalizability. While these constraints were necessary for practical implementations, exploring more flexible constraint formulations could have further enhanced the method's adaptability to different metro systems and operational conditions.

Authors in [220] proposed an advanced scheduling algorithm for multitask environments using a combination of PPO and an optimal policy characterization. The main focus was to optimize the scheduling of multiple tasks across multiple servers, taking into account random task arrivals and renewable energy generation. One of the primary strengths of this paper was its innovative combination of PPO with a priority rule, the Earlier Deadline and Less Demand First (ED-LDF). This rule prioritized tasks with earlier deadlines and lower demands, which was shown to be optimal under heavy traffic conditions. The integration of ED-LDF with PPO effectively reduced the dimensionality of the action space, making the algorithm scalable and efficient even in large-scale settings. This reduction in complexity was crucial for practical applications where the number of tasks and servers could have been substantial. However, the complexity of implementing the proposed method might have posed challenges. The integration of the ED-LDF rule with PPO required careful tuning and a deep understanding of both RL and optimal scheduling principles. This complexity could have limited the accessibility of the method for practitioners who might not have had advanced expertise in these areas. Additionally, the reliance on heavy-traffic assumptions for the optimality of the ED-LDF rule might have limited the generalizability of the approach to all scheduling environments. Real-world scenarios could have presented varying traffic conditions that did not always align with these assumptions. The paper's focus on renewable energy as a factor in the scheduling decision was another strength, as it aligned with the growing importance of sustainability in computing operations. However, the paper could have benefited from further exploration of how variations in renewable energy availability impacted the scheduling performance. This aspect was crucial for real-world applications where renewable energy sources could have been highly variable.

An innovative approach to enhancing image captioning models using PPO was designed by the authors in [221]. The authors aimed to improve the quality of generated captions by incorporating PPO into the phase of training, specifically targeting the optimization of scores. The study explored various modifications to the PPO algorithm to adapt it effectively for the image captioning task. A significant strength of the paper was its inte-

gration of PPO with image captioning models, which traditionally relied on VPG methods. The authors argued that PPO could provide better performance due to its ability to enforce trust-region constraints, thereby improving sample complexity and ensuring stable policy updates. This was particularly important for image captioning, where maintaining high-quality training trajectories was crucial. The authors' experimentation with different regularization techniques and baselines was another strong point. They found that combining PPO with dropout decreased performance, which they attributed to increased KL-divergence of RL policies. This empirical observation was critical as it guided future implementations of PPO in similar contexts. Furthermore, the adoption of a word-level baseline via MC estimation, as opposed to the traditional sentence-level baseline, was a noteworthy innovation. This approach was expected to reduce the variance of policy gradient estimators more effectively, contributing to improved model performance. While the results were promising, they were primarily validated on the MSCOCO dataset. Further validation on other datasets and in real-world applications would have been beneficial to assess the generalizability and robustness of the approach. The paper could have also benefited from a more detailed discussion on the impact of different hyperparameter settings and the specific configurations used in the experiments. Providing this information would have enhanced the reproducibility of the study and allowed other researchers to build on the authors' work more effectively.

In [222], a centralized coordination scheme for CAVs at intersections without traffic signals was developed. The authors introduced the Model Accelerated PPO (MA-PPO) algorithm, which incorporated a prior model into the PPO algorithm to enhance sample efficiency and reduce computational overhead. One of the significant strengths of this paper was its focus on improving computational efficiency, a major challenge in centralized coordination methods. Traditional methods, such as MPC, were computationally demanding, making them impractical for real-time applications with a large number of vehicles. By using MA-PPO, the authors significantly reduced the computation time required for coordination, achieving an impressive reduction to 1/400 of the time needed by MPC. This efficiency gain was crucial for real-time deployment in busy intersections. A limitation was the focus on a specific intersection scenario. While the four-way single-lane intersection was a common setup, real-world intersections could have varied significantly in complexity and traffic patterns. Future work could have explored the applicability of MA-PPO to more complex intersection scenarios and different traffic conditions to ensure broader generalizability.

The application of PPO in developing a DRL controller for the nonlinear attitude control of fixed-wing UAVs was explored in [223]. The study presented a proof-of-concept controller capable of stabilizing a fixed-wing UAV from various initial conditions to desired

roll, pitch, and airspeed values. One of the primary strengths of the paper was its innovative use of PPO for UAV attitude control. PPO was known for its stability and efficient policy updates, making it well-suited for complex control tasks like UAV attitude stabilization. The choice of PPO over other RL algorithms was justified by its robust performance and low computational complexity, which were crucial for real-time control applications. The authors also highlighted the practical advantages of using PPO, such as its hyperparameter robustness across different tasks. One of the limitations was the reliance on a single UAV model and specific aerodynamic coefficients for validation. While the Skywalker-X8 was a popular fixed-wing UAV, the generalizability of the proposed approach to other UAV models with different aerodynamic characteristics remained to be explored. Future work could have benefited from testing the PPO-based controller on a wider range of UAV models to ensure broader applicability.

The use of PPO to control the position of a quadrotor was investigated in [224]. The primary goal was to achieve stable flight control without relying on a predefined mathematical model of the quadrotor's dynamics. One of the major strengths of this paper was its application of PPO in the context of quadrotor control. By using PPO, the authors ensured that the control policy updates remained stable and efficient, which was crucial for real-time applications like quadrotor control. The choice of PPO over other RL algorithms was well justified due to its robustness and low computational complexity. The authors' approach to utilizing a stochastic policy gradient method during training, which was then converted to a deterministic policy for control, was another notable strength. This strategy ensured efficient exploration during training, allowing the quadrotor to learn a robust control policy. The use of a simple reward function that focused on minimizing the position error between the quadrotor and the target further added to the efficiency of the training process. One limitation was the reliance on specific initial conditions and a fixed simulation environment. While the authors showed that the PPO-based controller could recover from harsh initial conditions, the generalizability of the method to different quadrotor models and varying environmental conditions remained to be explored. Future work could have benefited from testing the controller on a wider range of scenarios and incorporating additional environmental factors to ensure broader applicability.

Authors in [225] addressed the development of an intelligent lane change strategy for autonomous vehicles using PPO. This approach uses PPO to manage lane change maneuvers in dynamic and complex traffic environments, focusing on enhancing safety, efficiency, and comfort. The authors' design of a comprehensive reward function that considered multiple aspects of lane change maneuvers was one of the strengths. The reward function incorporated components for safety (avoiding collisions and near-collisions), efficiency (minimizing travel time and aligning with desired speed and position), and comfort (reducing lateral

and longitudinal jerks). This multi-faceted approach ensured that the learned policy optimized for a holistic driving experience, balancing the often competing demands of these different aspects. The inclusion of a safety intervention module to prevent catastrophic actions was a particularly noteworthy feature. This module labeled actions as "catastrophic" or "safe" and could replace potentially dangerous actions with safer alternatives, enhancing the robustness of the learning process. This safety-centric approach addressed a critical concern in applying DRL to real-world autonomous driving tasks, where safety was paramount. However, the complexity of implementing PPO for lane change maneuvers posed challenges. The need for continuous training and fine-tuning of parameters could have been resource-intensive and might not have been feasible for all developers or organizations, acknowledging that it is true for some other algorithms.

We now turn our attention to the final group of methods, actor-critic methods. We begin by introducing the general actor-critic approach, which combines both value-based and policy-based methods.

3.4 Actor-Critic Methods

Actor-critic methods combine value-based and policy-based approaches. These methods consist of two key components: the actor, which selects actions based on a policy, and the critic, which evaluates the actions based on their value function. By providing feedback on the quality of the actions taken, the critic helps guide the actor in updating the policy. This collaboration enables more stable and efficient learning, addressing some of the limitations found in pure policy or value-based approaches [226, 227].

In the next subsection, we will first introduce two major variants of actor-critic methods, namely Asynchronous Advantage Actor-Critic (A3C) and Advantage Actor-Critic (A2C), and then analyze their various applications.

3.4.1 A3C & A2C

The A2C algorithm is a synchronous variant of the A3C algorithm, which was introduced by [228]. A2C maintains the key principles of A3C but simplifies the training process by synchronizing the updates of multiple agents, thereby leveraging the strengths of both actor-critic methods and advantage estimation. The actor-critic architecture combines two primary components in both algorithms: the actor, which is responsible for selecting actions, and the critic, which evaluates the actions by estimating the value function. The

actor updates the policy parameters in a direction that is expected to increase the expected reward, while the critic provides feedback by computing the TD error. This integration allows for more stable and efficient learning compared to using Actor-only or critic-only methods [229]. Advantage estimation is a technique used to reduce the variance of the policy gradient updates. The advantage function $A(s, a)$ represents the difference between the action-value function $Q(s, a)$ and the value function $V(s)$:

$$A(s, a) = Q(s, a) - V(s). \tag{3.18}$$

By utilizing the advantage function, A2C focuses on actions that yield higher returns than the average, which aids in making more informed policy updates [3]. In contrast to A3C, where multiple agents update the global model asynchronously, A2C synchronizes these updates. Multiple agents operate in parallel environments, gathering experiences and computing gradients, which are then aggregated and used to update the global model in a synchronized manner. This synchronization simplifies the implementation and avoids issues associated with asynchronous updates, such as non-deterministic behavior and the potential overwriting of gradients.

The A3C algorithm operates as follows (see Fig. 3.5): Based on Alg. 24, first, the parameters of the policy network (actor) θ and the value network (critic) ϕ are initialized (line 1). Then, multiple agents are run in parallel, each interacting with its own copy of the environment (lines 2-14). Each agent independently collects a batch of experiences (s_t, a_t, r_t, s_{t+1}) (lines 5-9). For each agent, the advantage is computed using the collected experiences. Subsequently, the gradients of the policy and value networks are calculated using the advantage estimates. **Finally, each agent independently updates the global model parameters θ and ϕ asynchronously using the computed gradients (line 13).**

The A2C algorithm, on the other hand, operates as follows: Based on Alg. 25, first, the parameters of the policy network (actor) θ and the value network (critic) ϕ are initialized (line 1). Then, multiple agents are run in parallel, each interacting with its own copy of the environment (line 4). Each agent collects a batch of experiences (s_t, a_t, r_t, s_{t+1}) (line 10). For each agent, the advantage is computed using the collected experiences. Subsequently, the gradients of the policy and value networks are calculated using the advantage estimates (lines 13-15). **Finally, the gradients from all agents are aggregated and used to update the global model parameters θ and ϕ (lines 17-18).**

Overview of A3C applications in the literature: [232] introduced a scheduler named MOPTSA3C, which prioritized tasks and virtual machines based on various factors, such as

Figure 3.5: The A3C architecture [228], implements a global network based on the actor-critic framework. It employs n worker threads, each sharing the same network structure as the global network. These threads interact independently with their respective environments and contribute to training by updating the global network parameters. Periodically, the parameters of the worker threads are synchronized with those of the global network [122].

task length, runtime processing capacities, and electricity unit costs. This approach aimed to optimize makespan, resource utilization, and resource cost using an enhanced A3C. One of the significant strengths of this paper was its comprehensive multi-objective approach. By addressing multiple objectives simultaneously, including minimizing makespan, optimizing resource utilization, and reducing resource costs, the proposed scheduler ensured a balanced and efficient task scheduling process in multi-cloud environments. The use of an improved A3C algorithm enhanced the robustness and efficiency of the scheduling process. The asynchronous nature of A3C allowed for parallel training and faster convergence, which was crucial for dynamic and large-scale cloud environments. The scheduler's ability to prioritize tasks and virtual machines based on multiple factors led to more informed and effective scheduling decisions. Cloud environments were inherently dynamic and could exhibit unpredictable changes in workload, resource availability, and cost structures. The proposed method might have needed further enhancements to adapt to these real-time variations effectively and ensure robust performance under fluctuating conditions. While

Algorithm 24 A3C [230]

```
Initialize actor and critic networks with random weights
for each episode ∈ [1, n] do
    Download weights from the headquarters to each AC
    for each AC do
        Initialize the random state s_0
        for each t ∈ [1, k] do
            Select action a_t from the actor network
            Execute action a_t and observe reward r_t from the critic network and next state s_t
            Update the actor network parameters
        end for
        Update the critic network parameters
    end for
    Upload weights of each AC network to the headquarters
end for
```

the paper presented a significant advancement in task scheduling for cloud computing, addressing the computational complexity, scalability, and real-world implementation challenges would have been crucial for its practical adoption and effectiveness in diverse cloud environments.

A latency-oriented cooperative caching strategy using A3C was proposed in [233]. This approach aimed to minimize average download latency by predicting content popularity and optimizing caching decisions in real time, thereby improving user experience in fog radio access networks. One strength was the comprehensive evaluation of the proposed method through extensive simulations. The authors compared their A3C-based approach with several baseline algorithms, including greedy caching, random caching, and a cooperative caching strategy that did not consider user equipment caching capacity. The results demonstrated significant reductions in average download latency, highlighting the effectiveness of the proposed strategy in optimizing caching performance. Another limitation was the reliance on accurate and timely content popularity predictions. The effectiveness of the caching strategy depended heavily on the accuracy of these predictions. In real-world applications, user preferences and content popularity could change unpredictably, and any deviations from the predicted values could negatively impact the performance of the caching algorithm. Additionally, the assumption that users did not request the same content repeatedly might not have held true in all scenarios, potentially affecting the reliability of the popularity prediction model.

Authors in [234] presented a comprehensive approach to optimizing resource allocation

Algorithm 25 A2C [231]

```
Initialize policy network (actor) parameters θ and value network (critic) parameters φ
Set number of parallel agents N
repeat
    for each agent i ← 1 to N in parallel do
        Get initial state s_i
        Initialize local episode storage for agent i
        for each step t do
            Sample action a_i from policy π_θ(a_i|s_i)
            Execute a_i, observe reward r_i and next state s'_i
            Store (s_i, a_i, r_i, s'_i) in local storage for agent i
            s_i ← s'_i
        end for
        Compute advantage estimates Â_i for agent i
        and policy gradient ∇_θ L_PG(θ) for agent i
        and value loss ∇_φ L_V(φ) for agent i
    end for
    Aggregate gradients from all agents
    Update global actor parameters θ and critic parameters φ using aggregated gradients
until convergence or maximum steps reached
```

and pricing in Mobile Edge Computing (MEC)-enabled blockchain systems using the A3C algorithm. The study's strengths lay in its innovative integration of blockchain with MEC to enhance resource management. The A3C algorithm's capability to handle both continuous and high-dimensional discrete action spaces made it well-suited for the dynamic nature of MEC environments. The use of prospect theory to balance risks and rewards based on miner preferences added a nuanced understanding of real-world scenarios. The results demonstrated that the proposed A3C-based algorithm outperformed baseline algorithms in terms of total reward and convergence speed, indicating its effectiveness in optimizing long-term performance. However, the paper had several limitations. The reliance on a specific MEC server configuration and a fixed number of mobile devices might have limited the generalizability of the findings to other settings with different configurations. The assumption that all validators were honest simplified the model but might not have reflected real-world blockchain environments where malicious actors could have existed. The additional complexity introduced by the collaborative local MEC task processing mode might have increased the computational overhead, potentially affecting the

scalability of the proposed solution. Moreover, the paper did not address the potential impact of network latency and varying network conditions on the performance of the A3C algorithm, which could have been significant in practical deployments.

In [235], the authors explored the application of A3C to create a cognitive agent designed to help Alzheimer's patients play chess. The primary goal was to enhance cognitive skills and boost brain activity through chess, a game known for its cognitive benefits. One of the key strengths of the paper was its innovative approach to leveraging A3C to assist individuals with cognitive disabilities in playing chess. The cognitive agent provided real-time assistance by suggesting offensive and defensive moves, thereby helping players improve their strategies and cognitive abilities. This approach addressed the gap in traditional AI chess opponents, which did not educate players on strategies and tactics. The cognitive agent relied on accurate feedback for consistency. Agent effectiveness was heavily dependent on its ability to provide relevant and timely suggestions. The agent needed to accurately interpret the player's intentions and provide appropriate feedback in real-world applications. Users who had difficulty navigating digital interfaces might also have experienced accessibility issues due to the agent's reliance on digital interfaces. Another concern was scalability. A controlled environment showed promising results, but the system might not have been able to handle a broader range of cognitive impairments. Additional users and interactions could have introduced additional complexity, making it difficult to maintain performance.

Researchers in [236] introduced a novel approach to autonomous valet parking using a combination of PPO and A3C. This method aimed to address the control errors due to the non-linear dynamics of vehicles to optimize parking maneuvers. An important strength of this study was the use of the A3C-PPO algorithm. Combining the advantages of both PPO and A3C, this hybrid approach resulted in a more stable and efficient learning process. A3C's asynchronous nature allowed parallel training, which sped convergence and improved state-action exploration. In addition, PPO prevented drastic changes from destabilizing the learning process by limiting the magnitude of policy updates. Incorporating manual hyperparameter tuning further optimized the training process, resulting in better rewards. One of the limitations was the reliance on specific assumptions about the environment and sensor accuracy. The effectiveness of the proposed method depended on the accurate detection and interpretation of the surroundings by sensors such as cameras and LiDAR. Any inaccuracies or deviations in sensor data could have impacted the performance and robustness of the algorithm. In real-world applications, environmental conditions such as lighting, weather, and obstacles could vary significantly, and ensuring reliable sensor performance under these conditions was crucial. Additionally, the study did not address the potential impact of network latency and communication issues between the vehicle and

the central control system, which could have affected the real-time decision-making process. Ensuring robust communication and minimizing latency was critical for the practical implementation of autonomous valet parking systems.

An advanced approach for optimizing content caching in 5G networks using A3C was proposed in [237]. This method aimed to minimize the total transmission cost by learning optimal caching and sharing policies among cooperative Base Stations (BSs) without prior knowledge of content popularity distribution. One of the main strengths of this paper was the use of the A3C algorithm, which leveraged the asynchronous nature of multiple agents to achieve faster convergence and reduce time correlation in learning samples. The algorithm's ability to operate with multiple environment instances in parallel enhanced computational efficiency and significantly improved the learning process. By considering cooperative BSs that could have fetched content from neighboring BSs or the backbone network, the proposed method effectively reduced data traffic and transmission costs in 5G networks. The empirical results demonstrated the superiority of the A3C-based algorithm over classical caching policies, such as least recently used, least frequently used, and adaptive replacement cache, showcasing lower transmission costs and faster convergence rates. One limitation of this study was the reliance on accurate and timely updates of content popularity distributions. While the paper assumed that content popularity followed a Zipf distribution and varied over time, the accuracy of these assumptions could have significantly impacted the performance of the caching algorithm. In real-world applications, user preferences and content popularity could have changed unpredictably, and any deviations from the assumed distribution could have affected the effectiveness of the caching policy. Ensuring robust performance under varying content popularity distributions was crucial for the practical implementation of the proposed method.

Overview of A2C applications in the literature: In [238], the authors introduced an innovative approach to low-latency task scheduling in edge computing environments, addressing several significant challenges inherent in such settings. The primary focus of the paper was on integrating a hard attention mechanism with the A2C algorithm to enhance task scheduling efficiency and reduce latency. The strengths of the Hard Attention Mechanism and Advantage Actor-Critic (HA-A2C) method lay in its ability to significantly reduce task latency by approximately 40% compared to the DQN method. The hard attention mechanism employed by HA-A2C was particularly effective in reducing computational complexity and increasing efficiency, allowing the model to process tasks more quickly. Additionally, the method showcased improved scalability, maintaining low task latency even as the number of tasks increased. The use of the A2C algorithm, which combined policy gradient and value function estimates, enhanced the stability and effective-

ness of the policy network, further contributing to the overall performance of the model. However, there were some limitations to the HA-A2C approach. One notable weakness was the potential complexity of implementing the hard attention mechanism in real-world scenarios, where the dynamic and heterogeneous nature of edge environments might have posed additional challenges. Furthermore, while the HA-A2C method outperformed other DRL methods in terms of latency, it might have still faced difficulties in scenarios with extremely high-dimensional states and action spaces, where further optimization might have been necessary. Another consideration was the reliance on accurate and timely data for effective attention allocation, which might not have always been feasible in practical applications.

A task scheduling mechanism in cloud-fog environments that leveraged the A2C algorithm was presented in [239]. This approach aimed to optimize the scheduling process for scalability, reliability, trust, and makespan efficiency. One of the significant strengths of the paper was the holistic approach it took toward task scheduling in heterogeneous cloud-fog environments. The use of the A2C algorithm was particularly effective in handling the dynamic nature of task scheduling, as it allowed the scheduler to make real-time decisions based on the current state of the system. By dynamically adjusting the number of virtual machines according to workload demands, the proposed scheduler ensured efficient resource utilization, which was crucial for maintaining system performance under varying conditions. One limitation was the reliance on specific system parameters and assumptions about the environment. The proposed method assumed accurate estimation of factors such as task priorities and virtual machine capacities, which might not have always been feasible in practical applications. Deviations from these assumptions could have impacted the performance and robustness of the scheduling algorithm. Further research could have explored the robustness of the proposed approach under more relaxed assumptions and in diverse real-world scenarios.

Authors in [240] introduced an A2C-learning-based framework for optimizing beam selection and transmission power in mmWave networks. This approach aimed to improve energy efficiency while maintaining coverage in dynamic and complex network environments. A notable strength of the paper was its innovative application of the A2C algorithm for the joint optimization of beam selection and transmission power. This dual optimization was particularly effective in addressing the significant challenge of energy consumption in mmWave networks. By leveraging A2C, the proposed method dynamically adjusted beam selection and power levels based on the current state of the network, which was represented by the Signal-to-Interference-plus-Noise Ratio (SINR) values. The use of A2C ensured stable and efficient learning through policy gradients and value function approximations, making it suitable for real-time applications. One of the limitations was the

assumption of specific system parameters and environmental conditions. The method assumed an accurate estimation of SINR values and predefined beam angles, which might not always have been feasible in practical applications. Deviations from these assumptions could have impacted the performance and robustness of the optimization algorithm. Future research could have explored the robustness of the proposed method under more relaxed assumptions and in diverse real-world scenarios.

Authors explored the use of the A2C algorithm to estimate the power delay profile (PDP) in 5G New Radio environments in [241]. This approach aimed to enhance channel estimation performance by leveraging DRL techniques. A notable strength of this paper was its innovative application of the A2C algorithm to the problem of PDP estimation. By framing the estimation problem within an RL context, the proposed method directly targeted the minimization of Mean Square Error in channel estimation, rather than aiming to approximate an ideal PDP. This pragmatic approach allowed the algorithm to adapt to the inherent approximations and imperfections in practical channel estimation processes, leading to improved performance. However, the complexity of implementing the A2C algorithm in real-world scenarios posed challenges. The need for extensive training and parameter tuning required significant computational resources and expertise in RL, which might not have been readily available in all settings. Additionally, while the simulation results were promising, further validation in real-world deployments was necessary to fully assess the robustness and practicality of the proposed approach. Real-world environments could have introduced additional challenges, such as varying traffic patterns and hardware constraints, which were not fully captured in simulations.

The application of various actor-critic algorithms to develop a trading agent for the Indian stock market was investigated in [242]. The study evaluated the performance of PPO, DDPG, A2C, and Twin Delayed DDPG (TD3) algorithms in making trading decisions. One of the primary strengths of this paper was its comprehensive approach to evaluating multiple actor-critic algorithms in a real-world financial trading context. By considering different algorithms, the authors provided a broad perspective on the effectiveness of it in stock trading. The use of historical stock data from the Yahoo Finance API, covering a substantial period (2006-2021), ensured that the models were tested against diverse market conditions, enhancing the robustness and reliability of the results. The study's focus on a single market (the Indian stock market) might have limited the generalizability of the findings. Future research could have explored the application of these algorithms in different financial markets to ensure broader applicability.

The authors presented an innovative framework for optimizing task segmentation and parallel scheduling in edge computing networks using the A2C algorithm in [243]. The approach focused on minimizing total task execution delay by splitting multiple computing-

intensive tasks into sub-tasks and scheduling them efficiently across different edge servers. A key strength of this paper was its holistic approach to task segmentation and scheduling. By jointly optimizing both processes, the proposed method ensured that tasks were not only divided efficiently but also assigned to the most suitable edge servers for processing. This joint optimization was crucial in dynamic edge computing environments, where both computation capacity and task requirements could have varied significantly over time. The use of A2C allowed the system to adapt to these changes in real-time, enhancing overall system performance. The authors' method of decoupling the complex mixed-integer non-convex problem into more manageable sub-problems was another strength. By first addressing the task segmentation problem and then tackling the sub-tasks parallel scheduling, the paper presented a structured and logical approach to solving the optimization challenge. The introduction of the optimal task split ratio function and its integration into the A2C algorithm further enhanced the efficiency and effectiveness of the proposed solution.

Researchers in [244] presented an innovative approach to enhancing multi-UAV obstacle avoidance using A2C combined with an experience-sharing mechanism. This method aimed to optimize obstacle avoidance strategies in complex, dynamic environments by sharing positive experiences among UAVs to expedite the training process. One of the key strengths of this paper was the introduction of the experience-sharing mechanism to the A2C algorithm. This mechanism significantly enhanced the efficiency and robustness of the training process by allowing UAVs to share positive experiences. This collective learning approach accelerated the convergence of the algorithm, enabling UAVs to quickly learn effective obstacle avoidance strategies. The experience-sharing mechanism was particularly valuable in multi-agent systems, where individual agents could have benefited from the knowledge gained by others, leading to faster and more robust learning. However, the experience-sharing mechanism relied on consistent and reliable communication between UAVs. In practical applications, communication constraints and network reliability issues could have significantly impacted the effectiveness of this mechanism. Inter-UAV communication latency and packet loss could have led to outdated or incomplete information being shared, thereby reducing the overall efficiency of the learning process. Also, the method assumed a certain level of accuracy in modeling the environment and the dynamic obstacles within it. Any deviations from these assumptions, such as unexpected changes in obstacle behavior or environmental conditions, could have affected the performance and robustness of the algorithm. Real-world environments were inherently unpredictable, and the algorithm must have been tested extensively in diverse scenarios to ensure its reliability.

A robust approach to visual navigation using an actor-critic method enhanced with Generalized Advantage Estimation (GAE) was developed by authors in [245]. This method demonstrated significant strengths in terms of learning efficiency and stability, as well as

effective navigation in complex environments like ViZDoom. One major strength of this approach was its ability to rapidly converge and achieve high performance in both basic and complex visual navigation tasks. By employing the A2C method with GAE, the algorithm reduced variance in policy gradient estimates, leading to more stable learning. This was particularly evident in the ViZDoom health gathering scenarios, where the A2C with GAE agent achieved the highest scores with lower variance compared to other methods. Additionally, the use of multiple processes in the A2C method significantly reduced training time, making it more efficient than traditional DQN approaches. However, the method also had notable limitations. One significant drawback was the high computational cost associated with using multiple processes for training, which might not have been feasible in resource-constrained environments. Furthermore, while the approach performed well in the tested ViZDoom scenarios, its generalizability to other, more diverse environments remained uncertain without further validation. The reliance on visual inputs also presented challenges in environments with varying lighting conditions or visual obstructions, which were not extensively tested in this study. Another limitation was the potential for overfitting to specific task environments. The training setup in controlled ViZDoom scenarios might not have fully captured the complexities of real-world navigation tasks, where environmental dynamics are less predictable. Thus, while the A2C with GAE approach showed promise, its applicability to a broader range of visual navigation tasks would have benefited from additional research and testing in more varied and less controlled environments.

In the following subsection, we will discuss the Deterministic Policy Gradient (DPG) algorithm, which addresses the challenges associated with **continuous action spaces**.

3.4.2 Deterministic Policy Gradient (DPG)

DPG addresses the challenges associated with continuous action spaces and offers significant improvements in sample efficiency over stochastic policy gradient methods. Traditional policy gradient methods in RL use stochastic policies, where the policy $\pi_\theta(a|s)$ is a probability distribution over actions given a state, parameterized by θ. These methods rely on sampling actions from this distribution to compute the policy gradient, which can be computationally expensive and sample inefficient, especially in high-dimensional action spaces [3, 246]. In contrast, the DPG algorithm uses a deterministic policy, denoted by $\mu_\theta(s)$, which directly maps states to actions without involving any randomness. The policy gradient theorem for deterministic policies shows that the gradient of the expected return with respect to the policy parameters can be computed as [246]:

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)\big|_{a=\mu_\theta(s)}\right] \tag{3.19}$$

where $Q^\mu(s,a)$ is the action-value function under the deterministic policy μ_θ, and ρ^μ is the discounted state visitation distribution under μ_θ.

By using an off-policy learning approach, DPG ensures adequate exploration while learning a deterministic target policy. A behavior policy, typically a stochastic policy, is employed to generate exploratory actions and collect experiences. The gradients obtained from these experiences are then used to update the deterministic policy. Since the same experiences can be reused to improve the policy in this non-policy setting, the data collected can be utilized more efficiently than in a policy-based setting [247]. The DPG algorithm is generally implemented within an actor-critic framework, where the actor represents the deterministic policy μ_θ, and the critic estimates the action-value function $Q^\mu(s,a)$. The critic is trained using TD learning to minimize the Bellman error:

$$\delta_t = r_t + \gamma Q^\mu(s_{t+1}, \mu_\theta(s_{t+1})) - Q^\mu(s_t, a_t) \tag{3.20}$$

where δ_t is the TD error, r_t is the reward, γ is the discount factor, and s_t, a_t are the state and action at time step t. The actor updates the policy parameters in the direction suggested by the critic:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \mu_\theta(s_t) \nabla_a Q^\mu(s_t, a_t)\big|_{a=\mu_\theta(s_t)} \tag{3.21}$$

where α is the learning rate for the actor. A variant of DPG, which is designed to handle continuous action spaces with the help of DL, is analyzed in the next subsection in detail.

3.4.3 Deep Deterministic Policy Gradient (DDPG)

The DDPG algorithm is an extension of the DPG method, designed to handle continuous action spaces effectively, introduced by [248]. DDPG leverages the power of DL to address the challenges associated with high-dimensional continuous control tasks [248]. The foundation of DDPG lies in the DPG algorithm. This approach contrasts with stochastic policy gradients, which sample actions from a probability distribution. The deterministic nature of DPG reduces the variance of gradient estimates and improves sample efficiency, making it suitable for continuous action spaces. DDPG employs an actor-critic architecture, where the actor network represents the policy $\mu(s|\theta^\mu)$ and the critic network estimates the action-value function $Q(s,a|\theta^Q)$. The actor network outputs a specific action for a given state, while the critic network evaluates the action by estimating the expected return. The policy gradient is computed using the chain rule:

$$\nabla_{\theta^\mu} J \approx \mathbb{E}_{s\sim\rho^\beta}\left[\nabla_a Q(s,a|\theta^Q)\big|_{a=\mu(s|\theta^\mu)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)\right] \tag{3.22}$$

where ρ^β denotes the state distribution under a behavior policy β. To stabilize learning and address the challenges of training with large, non-linear function approximators, DDPG incorporates two key techniques from the DQN algorithm [249]:

1. Replay Buffer: A replay buffer stores transitions (s_t, a_t, r_t, s_{t+1}) observed during training. By sampling mini-batches of transitions from this buffer, DDPG minimizes the correlations between consecutive samples, which stabilizes training and improves efficiency.

2. Target Networks: DDPG uses target networks for both the actor and critic, which are periodically updated with a soft update mechanism. The target networks provide stable targets for the Q-learning updates, reducing the likelihood of divergence:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{3.23}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{3.24}$$

   where $\tau < 1$ is the target update rate.

Exploration in continuous action spaces is crucial for effective learning. DDPG employs an exploration policy by adding noise to the actor's deterministic policy. An Ornstein-Uhlenbeck process [250] is typically used to generate temporally correlated noise, promoting exploration in environments with inertia.

The DDPG algorithm operates as follows: As shown in Alg. 26, first, the parameters of the actor network θ^μ and the critic network θ^Q are initialized [251] (line 1-2). Target networks for both the actor and critic are also initialized. Then, multiple agents interact with their respective environments, collecting transitions (s_t, a_t, r_t, s_{t+1}), which are stored in a replay buffer (lines 3-10). For each agent, the actor selects actions based on the current policy with added exploration noise. The critic network is updated using the Bellman equation to minimize the TD error (lines 11-13). The actor network is updated using the policy gradient derived from the critic (line 14). Periodically, the target networks are updated to slowly track the learned networks (line 15). In the next paragraphs, we analyze some of the papers in the literature that used DDPG.

Algorithm 26 DDPG [248]

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$.
Initialize replay buffer R.
for episode = 1 to M **do**
    Initialize a random process N for action exploration.
    Receive initial observation state s_1.
    for t = 1 to T **do**
        Select action $a_t = \mu(s_t|\theta^\mu) + N_t$ according to the current policy and exploration noise
        Execute action a_t and observe reward r_t and observe new state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in R
        Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$.
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$.
        Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \times \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

        Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

    end for
end for

Authors in [252] presented an innovative method for developing a missile lateral acceleration control system using the DDPG algorithm. This study reframed the autopilot control problem within the RL context, utilizing a 2-degrees-of-freedom nonlinear model of the missile's longitudinal dynamics for training. One strength was the incorporation of performance metrics, such as settling time, undershoot, and steady-state error, into the reward function. By integrating these key performance indicators, the authors ensured

that the trained agent not only learned to control the missile effectively but also adhered to desirable performance standards. This approach enhanced the practical applicability of the method, ensuring that the control system met operational requirements. The method's scalability to more complex scenarios and larger-scale implementations was another concern. The increased number of states and potential interactions in a real-world missile control system could have introduced additional complexities, making it challenging to maintain the same level of performance. Further research was needed to explore the scalability of the approach and develop mechanisms to manage the increased computational load.

A novel approach to fast charging lithium-ion batteries using a combination of a model-based state observer and a DRL optimizer, specifically the DDPG algorithm, was proposed in [253]. This method aimed to balance charging rapidity with the enforcement of thermal safety and degradation constraints. One of the notable strengths of this paper was its innovative application of the DDPG algorithm to the complex problem of fast charging lithium-ion batteries. By formulating a multi-objective optimization problem that included penalties for over-temperature and degradation, the authors effectively addressed the crucial aspects of battery safety and longevity. This approach ensured that the charging strategy not only accelerated the charging process but also maintained the battery within safe operating limits, thus extending its life. The method's scalability to different types of batteries and charging environments was a concern. While the results were promising for the specific battery model used in the study, the ability to generalize the approach to other battery types and configurations remained uncertain. The increased number of variables and potential interactions in more complex systems could have introduced additional complexities, making it challenging to maintain the same level of performance. Further research was needed to explore the scalability of the approach and develop mechanisms to manage the increased computational load.

Authors explored the application of DDPG to the problem of obstacle avoidance in the trajectory planning of a robot arm in [254]. The authors proposed using DDPG to plan the trajectory of a robot arm, ensuring smooth and continuous motion while avoiding obstacles. The rewards were specifically designed to overcome the convergence difficulties posed by multiple and potentially antagonistic rewards. One strength of this paper was the careful design of the reward function. By considering both the distance to the target and the proximity to obstacles, the authors ensured that the robot not only reached its goal efficiently but also avoided collisions. This multi-faceted reward structure addressed the convergence issues often encountered in RL tasks with multiple objectives. The empirical validation through simulations in the MuJoCo environment demonstrated the effectiveness of the proposed method, showing that the robot arm could successfully navigate to its

target while avoiding obstacles. The method's scalability to more complex scenarios and larger-scale implementations might have impeded the achieved performance. The increased number of states and potential interactions in a real-world robotic system could have introduced additional complexities, making it challenging to maintain the same level of performance. Further research was needed to explore the scalability of the approach and develop mechanisms to manage the increased computational load. Investigating the impact of various hyperparameters and network architectures on the performance of the DDPG algorithm could have provided deeper insights into optimizing the method for different robotic applications.

An enhanced version of DDPG, integrating a Long Short-Term Memory (LSTM) network-based encoder to handle dynamic obstacle avoidance for mobile robots in stochastic environments, is introduced in [255]. One of the notable strengths of this paper was the innovative combination of DDPG with LSTM. This hybrid approach allowed the robot to encode a variable number of obstacles into a fixed-length representation, which addressed the limitation of traditional DDPG that required a fixed number of inputs. By utilizing the LSTM network-based encoder, the proposed method could effectively process and integrate dynamic environmental information, enhancing the robot's adaptability to unpredictable scenarios. The reliance on accurate environmental sensing and real-time data processing was a potential limitation. The performance of the LSTM-based encoder and the overall DDPG framework heavily depended on the quality and accuracy of the sensor data. In real-world applications, factors such as sensor noise, varying environmental conditions, and communication delays could have affected the reliability and robustness of the system. Ensuring robust performance under diverse and unpredictable conditions remained a critical challenge.

DDPG, combined with prioritized sampling to optimize power control in wireless communication systems, specifically targeting Multiple Sweep Interference (MSI) scenarios, was designed in [256]. Prioritized sampling was another innovative aspect of the paper. By focusing on more valuable experiences during training, the algorithm accelerated the learning process and improved convergence speed. The empirical results showed that the DDPG scheme with prioritized sampling (DDPG-PS) outperformed the traditional DDPG scheme with uniform sampling and the DQN scheme. This was evident in various MSI scenarios, where the DDPG-PS scheme achieved better reward performance and stability. The scalability of the proposed method to more complex scenarios and larger-scale implementations was another concern. While the results were promising in simulated environments, the ability to handle a broader range of interference patterns and larger numbers of channels remained uncertain. The increased number of states and potential interactions could have introduced additional complexities, making it challenging to maintain the same level of

performance. Further research was needed to explore the scalability of the approach and develop mechanisms to manage the increased computational load.

Authors in [257] employed DDPG to model the bidding strategies of generation companies in electricity markets. This approach was aimed at overcoming the limitations of traditional game-theoretic methods and conventional RL algorithms, particularly in environments characterized by incomplete information and high-dimensional continuous state/action spaces. One significant strength was the ability of the proposed method to converge to the Nash Equilibrium even in an incomplete information environment. Traditional game-theoretic methods often required complete information and were limited to static games. In contrast, the DDPG-based approach could dynamically simulate repeated games and achieve stable convergence, demonstrating its robustness in modeling real-world market conditions. One limitation was the reliance on accurate modeling of market conditions and real-time data processing. The effectiveness of the proposed method depended heavily on the precision of the input data, such as nodal prices and load demands. In real-world applications, factors such as data inaccuracies, communication delays, and varying environmental conditions could have impacted the reliability and robustness of the system. Ensuring robust performance under diverse and unpredictable conditions remained a critical challenge that needed to be addressed. Additionally, the study assumed a specific structure for the NNs used in the actor and critic models. The performance of the algorithm could have been sensitive to the choice of network architecture and hyperparameters. A more systematic exploration of different architectures and their impact on performance could have provided deeper insights into optimizing the DDPG algorithm for electricity market modeling.

An advanced approach for resource allocation in vehicular communications using a multi-agent DDPG algorithm was studied in [258]. This method was designed to handle the dynamic and high-mobility nature of vehicular environments, specifically targeting the optimization of the sum rate of Vehicle-to-Infrastructure (V2I) communications while ensuring the latency and reliability of Vehicle-to-Vehicle (V2V) communications. One of the significant strengths was the formulation of the resource allocation problem as a decentralized Discrete-time and Finite-state MDP. This approach allowed each V2V communication to act as an independent agent, making decisions based on local observations without requiring global network information. This decentralization was crucial for scalability and real-time adaptability in high-mobility vehicular environments. One potential limitation was the reliance on the accurate and timely acquisition of CSI. In high-mobility vehicular environments, obtaining precise CSI could have been challenging due to fast-varying channel conditions. Any inaccuracies in CSI could have impacted the performance and robustness of the proposed resource allocation scheme. Ensuring robust performance under

diverse and unpredictable conditions remained a critical challenge. The last algorithm in the category of actor-critic methods to analyze is TD3, which is an enhancement of the DDPG algorithm designed to address the issues of overestimation bias. This algorithm is analyzed in detail in the next subsection.

3.4.4 Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 is an enhancement of the DDPG algorithm, introduced by [259], designed to address overestimation bias in function approximation within actor-critic methods. TD3 incorporates several innovative techniques to improve the stability and performance of continuous control tasks in RL. Overestimation bias arises when the value estimates for certain actions are consistently higher than their true values due to function approximation errors. This issue is well-recognized in value-based methods such as Q-learning [260]. In actor-critic methods, this bias can result in suboptimal policy updates and divergent behavior [259].

TD3 builds on the concept of double Q-learning, which mitigates overestimation bias by maintaining two separate value estimators and using the minimum of the two estimates for the target update [96]. This approach is adapted to the actor-critic framework by utilizing two critic networks, Q_{θ_1} and Q_{θ_2}, which are trained independently. The target value is computed as:

$$y = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \mu_{\theta'}(s')) \tag{3.25}$$

where $Q_{\theta'_i}$ are the target critic networks and $\mu_{\theta'}$ is the target actor network.

To further minimize error propagation from the critic to the actor, TD3 delays the policy updates in relation to the value updates. Specifically, the policy (actor) network is updated less frequently than the critic networks. This strategy ensures that the value estimates used for policy updates are more accurate and stable. The policy is updated every d iterations, where d is a hyperparameter typically set to 2 or higher [259].

To avoid overfitting to narrow peaks in the value estimates, TD3 introduces target policy smoothing. This technique adds noise to the target policy, promoting smoother value estimates and fostering more robust policy learning. The target value is computed with added noise:

$$y = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \mu_{\theta'}(s') + \epsilon) \tag{3.26}$$

where ϵ is clipped noise sampled from a Gaussian distribution. Based on Alg. 27, the TD3 algorithm operates as follows:

Algorithm 27 TD3 [259]

1: Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
2: Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
3: Initialize replay buffer $\mathcal{B}$
4: **for** $t = 1$ to T **do**
5: Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
6: Store transition tuple (s, a, r, s') in $\mathcal{B}$
7: Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
8: $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon$, $\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c)$
9: $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
10: Update critics $Q_{\theta_i} \leftarrow \arg\min_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
11: **if** $t \mod d = 0$ **then**
12: Update ϕ by the deterministic policy gradient: $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
13: Update target networks:

$$\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$$

$$\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$$

14: **end if**
15: **end for**

First, the parameters of the actor network θ and the critic networks θ_1 and θ_2 are initialized (line 1). Target networks for both the actor and critics are also initialized (line 2). Multiple agents interact with their respective environments, collecting transitions (s_t, a_t, r_t, s_{t+1}), which are stored in a replay buffer (line 6). The actor selects actions with added exploration noise (line 8). The critic networks are updated by minimizing the TD error using the clipped double Q-learning target (lines 9-10). The actor network is updated using the deterministic policy gradient, but only every d iterations (line 12). Periodically, the target networks are updated to slowly track the learned networks (line 13).

Authors in [261], as the first analyzed paper in the literature, showcased both strengths and limitations in applying the TD3 algorithm for energy management in Hybrid EVs (HEVs). One of the primary strengths of the paper was its innovative application of the TD3 algorithm, which enhanced training efficiency and stability over the previously used methods like Q-learning, DQN, and DDPG. The TD3 algorithm's use of two critic net-

works helped provide more stable training by mitigating the overestimation bias. The paper also demonstrated significant improvements in fuel economy and battery state-of-charge sustainability, which are crucial metrics for HEVs. By comparing the performance under various driving cycles, the authors provided comprehensive evidence of the TD3 algorithm's effectiveness in real-world scenarios. However, there were notable limitations. The implementation of TD3, while improving stability, introduced complexity in the training process, requiring careful tuning of hyperparameters to achieve optimal performance. The algorithm's reliance on extensive computational resources for training might have limited its practical applicability in scenarios with constrained resources. Additionally, the paper focused primarily on the simulation results without providing sufficient real-world testing to validate the algorithm's performance under actual driving conditions. This gap raised questions about the robustness of the proposed method when deployed in a real-world environment.

The application of the TD3 algorithm for the target tracking of UAVs was proposed in [262]. The authors integrated several enhancements into the TD3 framework to improve its performance in handling the high nonlinearity and dynamics of UAV control. A significant strength was the novel reward formulation that incorporated exponential functions to limit the effects of velocity and acceleration on the policy function approximation. This approach prevented deformation in the policy function, leading to more stable and robust learning outcomes. Additionally, the concept of multistage training, where the training process was divided into stages focusing on position, velocity, and acceleration sequentially, enhanced the learning efficiency and performance of the UAV in tracking tasks. However, the proposed method also had several limitations. The integration of a PD controller and the novel reward formulation added to the complexity of the training process. The scalability of the proposed method to more complex environments with a higher number of dynamic obstacles or more sophisticated UAV maneuvers was another concern. While the results were promising in the tested scenarios, the ability to handle a broader range of operational conditions and larger numbers of UAVs remained uncertain. The increased number of states and potential interactions could have introduced additional complexities, making it challenging to maintain the same level of performance.

A novel dynamic MsgA channel allocation strategy using TD3 to mitigate the issue of MsgA channel collisions in low earth orbit satellite communication systems is investigated in [263]. The paper's approach to dynamically pre-configuring the mapping relationship between PRACH occasions and PUSCH occasions based on historical access information was another strong point. This method allowed the system to adapt to changing access demands effectively, ensuring efficient use of available resources and reducing collision rates. The empirical results were impressive, demonstrating a 39.12% increase in access success

probability, which validated the effectiveness of the proposed strategy. The scalability of the proposed method to larger and more complex satellite networks was one of the concerns. While the results were promising in the tested scenarios, the ability to handle a broader range of interference patterns and a larger number of users remained uncertain. The increased number of states and potential interactions could have introduced additional complexities, making it challenging to maintain the same level of performance. Further research was needed to explore the scalability of the approach and develop mechanisms to manage the increased computational load.

A TD3-based method for optimizing voltage and reactive power in distribution networks with high penetration of Distributed Energy Resources (DERs) such as battery energy storage and solar photovoltaic units was investigated in [264]. The authors' approach of coordinating the reactive power outputs of fast-responding smart inverters and the active power of battery ESS enhanced the overall efficiency of the network. By carefully designing the reward function to ensure a proper voltage profile and effective scheduling of reactive power outputs, the method optimized both voltage regulation and power loss minimization. The results demonstrated that the TD3-based method outperformed traditional methods, such as local droop control and DDPG-based approaches, showing significant improvements in reducing voltage fluctuations and minimizing power loss in the IEEE 34- and 123-bus test systems. The scalability of the proposed method to larger and more complex distribution networks was another concern. While the results were promising in the tested IEEE 34- and 123-bus systems, the ability to handle a broader range of network configurations and a larger number of DERs remained uncertain. The increased number of states and potential interactions could have introduced additional complexities, making it challenging to maintain the same level of performance. Further research was needed to explore the scalability of the approach and develop mechanisms to manage the increased computational load.

Authors in [265] presented an innovative approach to quadrotor control, leveraging the TD3 algorithm to address stabilization and position tracking tasks. This study was notable for its application to handle the complex, non-linear dynamics of quadrotor systems. The authors' method of integrating target policy smoothing, twin critic networks, and delayed updates of value networks enhanced the learning efficiency and reduced variance in the policy updates. This comprehensive approach ensured that the quadrotor could achieve precise control in both stabilization and position tracking tasks. The empirical results demonstrated the effectiveness of the TD3-based controllers, showcasing significant improvements in achieving and maintaining target positions under various initial conditions. The scalability of the proposed method to more complex environments with dynamic obstacles and more sophisticated maneuvers was another concern. While the results were

promising in the tested scenarios, the ability to handle a broader range of operational conditions and larger-scale implementations remained uncertain. The increased number of states and potential interactions could have introduced additional complexities, making it challenging to maintain the same level of performance. Further research was needed to explore the scalability of the approach and develop mechanisms to manage the increased computational load.

A real-time charging navigation method for multi-autonomous underwater vehicles (AUVs) systems using the TD3 algorithm was designed in [266]. This method was designed to improve the efficiency of navigating AUVs to their respective charging stations by training a trajectory planning model in advance, eliminating the need for recalculating navigation paths for different initial positions, and avoiding dependence on sensor feedback or pre-arranged landmarks. The primary strength of this paper lies in its application of the TD3 algorithm to the multi-AUV charging navigation problem. By training the trajectory planning model in advance, the method significantly improved the real-time performance of multi-AUV navigation. However, the paper also had some limitations. One major limitation was the reliance on the accuracy of the AUV motion model and the assumptions made during its formulation. For instance, the model assumed constant velocity and neglected factors like water resistance and system delays, which could have affected the real-world applicability of the results. Moreover, the simulation environment used for training and testing might not have fully captured the complexities and variabilities of real underwater environments. Another potential limitation was the need for extensive computational resources for training the TD3 model, especially given the high number of training rounds (up to 6000) and the large experience replay buffer size.

Authors in [267] presented an advanced approach for Adaptive Cruise Control (ACC) using the TD3 algorithm. This method addressed the complexities of real-time decision-making and control in automotive applications. The authors carefully designed the reward function to consider the velocity error, control input, and additional terms to ensure stability and smooth driving behavior. This reward structure allowed the algorithm to learn an optimal policy that maintained safe distances between vehicles while adapting to changing traffic conditions. The empirical results demonstrated the effectiveness of the TD3-based ACC system in both normal and disturbance scenarios, highlighting its robustness and adaptability. The scalability of the proposed method to more complex driving scenarios and larger-scale implementations was another concern. While the results were promising in the tested scenarios, the ability to handle a broader range of operational conditions and larger numbers of vehicles remained uncertain. The increased number of states and potential interactions could have introduced additional complexities, making it challenging to maintain the same level of performance. Further research was needed to explore the scal-

ability of the approach and develop mechanisms to manage the increased computational load. Additionally, the study assumed a specific structure for the NNs used in the actor and critic models. The performance of the algorithm could have been sensitive to the choice of network architecture and hyperparameters. A more systematic exploration of different architectures and their impact on performance could have provided deeper insights into optimizing the TD3 algorithm for adaptive cruise control.

3.5 Analytical Experiments

In this section, an analytical experiment between multiple DRL algorithms on the Farama Highway environment [268] is given. This environment is a simulation framework designed for developing and evaluating RL algorithms in autonomous driving and traffic management scenarios. It provides a modular and highly customizable environment where agents can interact with complex highway scenarios, including multi-lane traffic, various vehicle types, and diverse driving behaviors. For this task in this section, three different modules are tested: highway-fast, intersection, and racetrack. Three different algorithms are used: DQN, A2C, and PPO. The ego vehicle, which is the agent, is distinguished by green color in the highway and intersection tasks and by light green in the racetrack task (See Fig. 3.6). In the highway task (Fig. 3.6a), the ego-vehicle is driving on a multilane highway populated with other vehicles. The agent's objective is to reach a high speed while avoiding collisions with neighboring vehicles. Driving on the right side of the road is also rewarded. The intersection task (Fig. 3.6b) is an intersection negotiation task with dense traffic. The last task, racetrack (Fig. 3.6c), is a continuous control task involving lane-keeping and obstacle avoidance [268].

DQN was selected as a baseline to compare the performance of other algorithms, as it is expected to underperform compared to more advanced methods. Additionally, DQN is suitable for discrete action spaces, making it a logical choice for tasks such as Racetrack, which involves continuous action spaces and poses an interesting challenge. This challenge is addressed later in the analysis. PPO was employed due to its widespread use and demonstrated superior performance compared to other algorithms. Furthermore, PPO is simpler to implement and tune, making it a practical choice. A2C was included as it serves as a strong competitor to PPO in terms of performance. Both PPO and A2C can handle continuous and discrete action spaces, providing flexibility across different tasks. The hyperparameters used for all algorithms were carefully selected to optimize performance. All algorithms were trained for at least 100,000 time steps to ensure sufficient learning. To analyze performance, at least 15 videos were generated for each algorithm, supplemented by Tensorboard for integrated monitoring. A consistent learning rate of $5e-4$ was adopted

(a) Highway task (b) Intersection task (c) Racetrack task

Figure 3.6: Farama Highway environment

after testing values ranging from $1e-4$ to $7e-4$. This value offered optimal performance. The discount factor γ was set to 0.9 to emphasize future rewards unless otherwise specified.

For DQN, a three-layer network using a multilayer perceptron architecture was chosen. The buffer size and batch size were set to 150,000 and 64, respectively. Increasing the buffer size beyond initial recommendations from Farama's documentation allowed for more stable learning by enabling the agent to utilize a larger pool of experiences. For PPO, the following hyperparameters were implemented:

```
model = PPO(
    "MlpPolicy",
    env,
    policy_kwargs=dict(net_arch=[dict(pi=[256, 256], vf=[256, 256])]),
    n_steps=batch_size * 12 // n_cpu,
    batch_size=batch_size,
    n_epochs=15,
    verbose=2,
    tensorboard_log="highway_ppo/",
)
```

The policy and value function architectures were deliberately kept small after testing alternative architectures, as no significant performance improvement was observed. For A2C, hyperparameters were chosen based on an analysis of performance metrics, including rewards and video outputs.

A comparative analysis was conducted across three tasks: Highway-fast-v0, Intersection-v0, and Racetrack-v0. In Highway-fast-v0, all algorithms performed well, but DQN exhibited more frequent lane changes compared to PPO and A2C. DQN also had a higher crash rate of approximately 20%, while PPO and A2C both achieved crash rates below 10%. PPO consistently outperformed A2C in this task. In Intersection-v0, PPO significantly outperformed A2C. DQN and A2C had higher crash rates, with DQN crashing 20-30% of the time and A2C exceeding this rate slightly. PPO proved to be the most effective, with a crash rate of less than 10%.

In the Racetrack task, which features continuous action spaces, PPO performed the best, with agents overtaking non-ego vehicles efficiently. A2C demonstrated slower speeds and occasional failures to overtake. DQN encountered significant implementation challenges as it is designed for discrete action spaces. Attempts to adapt the environment using `DiscreteMetaActions` and `DiscreteActions` were unsuccessful, leading to erratic agent behavior, including moving backward or circling a point. Adjusting low-level action controls improved DQN's functionality, but its performance remained suboptimal, with frequent crashes.

```
config = {
    "action": {
        "type": "DiscreteAction",
        "steering_range": [-0.7853981633974483, 0.7853981633974483],
        "longitudinal": False,
        "lateral": True,
        "target_speeds": [0, 5, 10],
    }
}
```

A2C and PPO were more effective than DQN in this scenario. The analysis highlights the task-specific nature of DRL and the importance of selecting and tuning algorithms based on the environment and requirements. PPO consistently outperformed other algorithms, while DQN struggled in more complex tasks due to its limitations with continuous action spaces.

In the subsequent chapters, various applications of RL will be introduced and explored in detail as part of the practical phase of this thesis.

3.6 Conclusion

Deep Reinforcement Learning has emerged as a transformative field, bridging the gap between RL and DL to address complex decision-making tasks. This chapter provides a structured analysis of DRL algorithms, spanning value-based, policy-based, and actor-critic methodologies, along with their key advancements, strengths, and limitations. DRL has achieved significant success across domains such as robotics, Intelligent Transportation Systems, and autonomous control, yet challenges such as computational resource demands, sensitivity to hyperparameter settings, and real-world scalability persist. Despite these limitations, the evolution of algorithms such as PPO, TD3 Policy Gradient, and other hybrid approaches have shown promising improvements in stability and convergence.

Chapter 4

Electric Vehicles Routing & Charging

This chapter presents a comparative analysis of RL and DRL techniques for optimizing reliable EV routing and charging in dynamic urban environments. By modeling the EV routing problem as a Markov Decision Process, our study accounts for real-world constraints such as battery levels, charging station availability, traffic conditions, and range anxiety. We evaluate three approaches: traditional Q-Learning, heuristic-augmented Q-Learning (TQL), and Deep Q-Learning (DQL) across simulated networks of varying scales. The experimental results demonstrate that DQL achieves faster convergence and consistently outperforms the other methods in minimizing travel time, improving distance efficiency, and maximizing cumulative rewards while ensuring more reliable route planning.

4.1 Introduction

Autonomous Electric Vehicles (AEVs) have emerged as a new and prominent approach and are expected to bring sustainable ways to alleviate traffic congestion, reduce energy consumption, and provide on-demand mobility services [269]. This surging demand, driven by cost-effectiveness, government regulations, reduction of emissions, and technological advancement, emphasizes the need to optimize such systems in a smart urban environment [270]. Despite intensive research on the solution to a large host of ethical, safety, and technical problems with AEV, publications that integrate autonomous charging and driving allow modest optimality in many cases due to complexity.

This paper emphasizes the need to identify the quickest reliable path for an AEV from its source to its destination while ensuring its battery charge remains above a recommended threshold to mitigate range anxiety. It takes into account where charging stations are

located and aims to optimize battery usage. Deciding when and where to charge is crucial, but so is determining how much charge is needed to complete the journey. This is especially important since charging an EV still takes more time than refueling with gasoline [271]. To address these challenges, the study proposes automating battery level management, route planning, and selecting the best charging stations based on various factors. The objective is to ensure that the AEV reaches its destination quickly while using its battery efficiently. More precisely, we dive into the dynamic nature of the environment in which AEVs operate, aiming to tackle the challenge of optimizing reliable routing and charging scenarios. Building upon the previous work [272], which explored heuristic methods for EV route optimization with charging considerations, and [273], where authors investigated the usage of Reinforcement Learning (RL) to solve the problem, we recognize the need for methodologies capable of adapting to temporal fluctuations to achieve optimal results. Our study aims to provide advanced RL and Deep Reinforcement Learning (DRL) techniques to improve the results achieved while adhering to the same constraints and objective.

The literature review indicates several existing approaches to EV routing and charging optimization. Zhang et al. [274] proposed a nonlinear programming model considering EV speed and loads for energy-efficient routing, while Hiermann et al. [275] emphasized distance-based approaches. Bruglieri et al. [276] utilized Mixed Integer Linear Programming (MILP) to serve fleets of EVs, and Michael et al. [277] employed tabu search heuristic methods. Machine learning techniques, as demonstrated by Masikos et al. [278] and Cauwer et al. [279], have also been utilized. Furthermore, studies by Bi et al. [280] and Cataldo et al. [281] address charging station coordination and battery charging strategies, respectively.

In the previous work, in [272], authors investigated the heuristic approaches. Later, in [273], they implemented a Q-learning approach and a new heuristic approach. A primary focus of our research is the integration of other RL and DRL algorithms to tackle the dynamic AEV routing problem.

Although Q-Learning performed decently in the previous work, we believe that other algorithms may perform better, and the need for comparison among various algorithms is felt. Thus, we develop other RL and DRL algorithms to compare with the Q-learning, as a baseline, to see the effects of the newly developed algorithms. The first algorithm is TQ-Learning, where we initialized the Q tables with the results derived from the Dynamic TERC2 algorithm [273]. Another algorithm used in this study is Deep Q-Learning (DQL), which uses Neural Networks [121]. We compared the results achieved by these algorithms to give a thorough comparison.

By conducting detailed evaluations across networks of different scales, we demonstrate that our new algorithms significantly surpass traditional Q-Learning in this context. Our

study uniquely compares three distinct RL methods, providing researchers with valuable insights into their performance and revealing more effective strategies for practical applications. Moreover, our work contributes to the ongoing advancement of optimizing electric vehicle routing in dynamic environments.

The remainder of this paper is organized as follows. Section 5.4.1 introduces the system model and describes the problem. Section 4.3 briefly explains the implemented heuristic method to solve the problem. The Reinforcement Learning approaches are explained in Section 5.4.3. Section 5.3.5 presents the performance evaluation.

4.2 System Model and Problem Description

The system model utilizes a weighted graph $G = (N, E)$ to represent the network, where N comprises nodes such as intersections and charging stations, and E consists of road segments connecting these nodes. Each node connection is given values derived from distance, travel duration, and battery status factors. As illustrated in Fig. 4.1, the starting and end points are shown by black nodes marked as S and D, respectively, whereas intersections and charging facilities are illustrated as blue and red nodes. The electric vehicle (EV) traverses from the source to the destination following the dotted arrow, with its energy depleting at a rate of $E_{\text{Depletion}}$ and replenishing when charging at a rate of E_{Charging}. To enhance battery longevity, the system enforces constraints on the minimum ($B_{\min}$) and maximum ($B_{\max}$) battery levels. Charging stations are identified by the set I_{Station}, and each road segment (i, j) has a fixed traversal time $T^{\text{Edge}}_{(i,j)}$. The primary objective is to determine the optimal route that maintains the battery level within the specified limits while minimizing the total travel time, which includes charging durations. Additionally, the EV must ensure that upon arrival at the destination, the battery level B_D satisfies $B_D = B_{\min} + E_{\text{toStation}}$, guaranteeing sufficient charge to reach the nearest charging station.

The routing challenge becomes even more intricate due to the ever-changing nature of the surroundings. Factors such as inconsistent waiting times at charging stations, varying traffic speeds, and unpredictable battery consumption rates all contribute to the complexity of the problem. For instance, an EV starting with a battery level of $B_S = 60\%$ may need to divert to a charging station to prevent the battery from dropping below the threshold $B_{\min} + E_{\text{toStation}}$, as demonstrated in the journey from node 1 to node 5 for recharging. This dynamic setting classifies the routing problem as NP-hard, analogous to the Traveling Salesman Problem (TSP). Unlike the classical TSP, the EV is not required to visit every charging station but must strategically select stops based on the necessity to maintain adequate charge levels. This added flexibility introduces additional complexity, necessitating

Figure 4.1: A visual representation of the system framework is provided, where intersections appear as blue nodes, and charging stations are marked in red. The source and destination points are denoted by S and D, respectively. Roads linking these nodes are depicted as edges. A dashed black arrow indicates the movement direction of an EV along a road, while each node is associated with the EV's estimated arrival time (T_i) and corresponding battery percentage level (B_i).

the development of advanced and innovative strategies to effectively address the routing and charging challenges. In subsequent sections, we will explore practical solutions tailored to these dynamic and state-dependent aspects of EV routing.

4.3 The Heuristic Approach: Dynamic TERC2

Alg. 28 details the dynamic TERC2 approach, which largely follows the TERC2 method described in [272], except for the modification applied when selecting the next charging station under conditions of insufficient battery capacity. The algorithm accepts a graph $G(N, E)$, the initial battery level B_s, the source node S, and the destination node D as inputs, and it computes the fastest route from S to D while addressing the constraints necessary for preserving battery life in a dynamic environment. Notably, line 3 of the algorithm updates edge-related information, such as traffic speed and battery consumption rate, thereby allowing the system to adjust its route decisions in response to changing conditions. This real-time update mechanism enhances the algorithm's ability to optimize

route selection, particularly when determining the nearest charging station from either the source or the destination.

Furthermore, the dynamic TERC2 method refines its process by computing a temporary time T_{temp} at line 17, which sums the time required to reach a charging station from the source (T_{temp1}) and the time from the charging station to the destination (T_{temp2}). Lines 18 to 20 then collectively identify the charging station that minimizes the overall travel time between the source and the destination. An important benefit of this approach is its capability to reroute the electric vehicle away from charging stations or destination points that initially appeared optimal but later proved inefficient due to evolving traffic conditions.

Algorithm 28 Dynamic TERC2

```
Input: Graph G(N, E), B_s, S, D
Output: P_T: The fastest path in a dynamic environment considering all constraints
  from S to D
T_D ← 0
function FindPath(S, D, B_S)
    E ← E_state
    P_T ← Dijkstra(S, D)
    T_P ← TimeOnPath(P_T)
    B_used ← BatteryUsedOnPath(P_T)
    if B_i − B_used ≥ B_min + E_toStation then
        T_D ← T_D + T_P
        return P_T
    else
        T_P ← L
        for each i ∈ I_Station do
            P_temp1 ← Dijkstra(S, i)
            P_temp2 ← Dijkstra(i, D)
            T_temp1 ← TimeOnPath(P_temp1)
            T_temp2 ← TimeOnPath(P_temp2)
            T_temp ← T_temp1 + T_temp2
            if T_P > T_temp then
                T_P ← T_temp
                S_new ← i
            end if
        end for
        B_new ← B_max
        T_D ← T_D + T_temp1 + G_i
        return P_temp1+ FindPath(S_new, D, B_new)
    end if
end function
```

4.4 Reinforcement Learning Approaches

Reinforcement Learning (RL) involves an agent interacting with its environment to develop a strategy that maximizes long-term rewards [1]. In our work, we frame the problem as

a Markov Decision Process (MDP) defined by the tuple (S, A, γ, P, R), where the set of states (S) represents a finite collection of conditions, with each state s_t corresponding to a specific time step t. The actions (A) available are the possible decisions the agent can take at any given state s_t, with each action denoted as $a_t \in A$. The discount factor (γ) is a parameter within the interval $[0, 1)$ that determines the relative importance of future rewards compared to immediate ones. The transition function (P), expressed as $P(s_{t+1} \mid s_t, a_t)$, specifies the probability of moving from the current state s_t to the next state s_{t+1} after taking action a_t. Lastly, the reward function (R), denoted as $R(s_t, a_t)$, assigns an immediate reward for executing action a_t in state s_t at time t.

Agent: Here, the agent is the EV itself. The EV is responsible for making decisions at every stage of its journey, choosing actions that help it navigate a changing environment while aiming to reduce its overall travel time.

State: The state space captures all the possible conditions the EV might encounter. Each state is characterized by several attributes: the EV's location within the road network (depicted by nodes), its current battery level, and the elapsed travel time. Additionally, reaching the destination is treated as entering a terminal state, which signals the end of the journey.

Actions: The available actions primarily involve moving the EV from one node to an adjacent one. Additionally, when the EV finds itself at a charging station, it has the extra option to recharge its battery to certain level(s).

Reward: The reward function is tailored to the goal of minimizing travel time. It assigns a negative value equal to the time consumed, which includes both the travel time between nodes and the time spent charging. Charging time is computed based on the difference between an optimal battery level (set at 80%) and the current battery level, expressed as

$$reward_{charging} = -(80 - current_{battery}) \tag{4.1}$$

Moreover, if the battery level falls below 20%, a significant penalty is imposed to encourage practices that extend battery longevity. Consequently, charging is limited to a maximum of 80%. By modeling our problem as an MDP, the EV, acting as the agent, learns to chart an optimal path through its environment. Through selecting actions that transition it to more advantageous states, the EV minimizes travel time while maximizing cumulative rewards. The details of the proposed RL and DRL approaches are presented below.

4.4.1 Q-Learning

The first used algorithm in this paper is Q-Learning, firstly introduced by [9], which is a model-free algorithm. Alg. 29 begins by initializing the Q-table for all state-action pairs to zero (line 1). This Q-table is fundamental to the Q-learning process as it stores the estimated rewards for executing specific actions from each state [1]. The algorithm is designed to work on a graph $G(N, E)$ with a defined start state S, a destination D, and designated charging stations C. Such a setup tailors the learning process to a constrained path planning problem, taking into account operational constraints such as battery charge and the availability of charging facilities.

For each of the K episodes (line 2), the algorithm sets the current node to the start state S (line 3) and records the initial path. Within each episode, a while-loop is executed until a termination condition is satisfied. At each iteration, an action is selected using an ϵ-greedy policy based on the current Q-table (line 6), which provides a balance between exploring new actions and exploiting known high-reward actions. The chosen action leads to the next state N_{next} (line 7), and the path is updated accordingly (line 8). A central aspect of this approach is the computation of the temporal difference error, δ, given by

$$\delta \leftarrow R + \gamma \times \max_{a'} Q(s', a') - Q(s, a) \tag{4.2}$$

(line 9). This error is then used to update the Q-value for the corresponding state-action pair using the formula

$$Q(s, a) \leftarrow Q(s, a) + \alpha \times \delta \tag{4.3}$$

The algorithm tracks the maximum Q-value change for convergence (line 10) and updates the current node to N_{next} (line 11). If the destination is reached or if the battery charge falls below zero (line 12) and the current reward exceeds the best recorded reward, the algorithm updates the best epoch results (lines 13-15) before exiting the loop. Finally, at the end of each episode, the algorithm calculates and records the epoch metrics, such as distance traveled and travel time (line 16). Additionally, the values of ϵ and α are decayed toward their minimum values (line 17) to gradually shift the behavior from exploration to exploitation. Once all K episodes have been executed, the algorithm returns the best epoch results containing the optimal path and its associated metrics (line 18). This method ensures that the final output is not only the most efficient path in terms of distance and time but also respects the constraints related to battery charge and charging station placement.

Algorithm 29 Q-Learning Algorithm

Require: Graph $G(N, E)$, Start State S, End State D, Charging Stations C
Ensure: Optimal Path P_T from S to D considering all constraints

```
Initialize Q(s, a) ← 0 for all state-action pairs
for k ← 1 to K do
    N_current ← S
    P ← [N_current]
    while True do
        Choose action a using ϵ-greedy policy based on Q
        Determine N_next based on action a
        Append N_next to path P
        δ ← R + γ × max_a' Q(s', a') − Q(s, a)
        Update Q-table: Q(s, a) ← Q(s, a) + α × δ
        Track maximum Q-value change for convergence
        N_current ← N_next
        if N_current = D or battery charge ≤ 0 then
            if current_reward > best_reward then
                Update best epoch results with current path and metrics
            end if
            break
        end if
    end while
    Calculate and record epoch metrics: distance & travel time
    Decay ϵ and α towards their minimum values
end for
return Best epoch results containing optimal path and metrics
```

4.4.2 TQ-Learning (TQL)

As some heuristic methods are implemented to solve this problem, we utilized the best one, Dynamic TERC2, to initialize the Q-tables of the Q-learning algorithm with its results. This leads to faster convergence, as it gives a good sub-optimal place for the algorithm instead of naive zeros, which has little advantage [3].

4.4.3 Deep Q-Learning (Deep Q-Networks)

The next proposed algorithm, shown in Alg. 30, merges Q-learning with Deep Learning's Neural Networks to learn control policies directly from the given input [121]. The algorithm begins by initializing the key components necessary for deep Q-learning, as shown in lines 1-4. In these initial steps, the policy network Q_θ and the target network Q_{θ^-} are both set up with random weights, and a replay buffer $\mathcal{D}$ with a fixed capacity is created to store state transitions. An initial battery charge is also assigned a value of 80, and essential hyperparameters, such as the exploration rate (ϵ), discount factor (γ), learning rate (α), and their decay rates, are established. These preparations ensure that the model has the structural and parameter foundations needed to learn the optimal policy for navigating the graph $G(N, E)$ while considering battery constraints and charging stations.

In the subsequent phase, encapsulated within the loop beginning at line 5, the algorithm iterates over a predetermined number of episodes K. At the start of each episode, the initial state S, battery level B, and path list are reset (lines 6-9). The core of the learning process is embedded in a while-loop (line 10) that continues until a terminal condition is met, either by reaching the destination D or by depleting the battery. Within each iteration, an action is chosen using an ϵ-greedy strategy: with probability ϵ a random valid action is selected, while with probability $1 - \epsilon$ the algorithm picks the action that maximizes the current Q-value from the policy network (lines 10-12). After executing the chosen action, the algorithm observes the resulting reward, the next state, and the updated battery charge, then stores this transition in the replay buffer (lines 13-14).

When the replay buffer has accumulated a sufficient number of samples (line 18), the algorithm performs a mini-batch update in line 19. For each sampled transition, it computes a target Q-value using the target network Q_{θ^-} by applying the Bellman equation, specifically, taking the maximum predicted Q-value for the next state and battery level, discounting it by γ, and considering whether the state is terminal (lines 20–22). The loss is then calculated as the mean squared error between these target Q-values and the Q-values predicted by the policy network (lines 23-24), and a gradient descent step is executed to update the network parameters accordingly (line 25). Additionally, to maintain training stability, the target network is periodically updated with the latest parameters from the policy network every C steps (line 26).

Algorithm 30 Deep Q-Learning Algorithm

Require: Graph $G(N, E)$, Start State S, End State D, Charging Stations C
Ensure: Optimal Path P_T from S to D

1: Initialize policy network Q_θ and target network Q_{θ^-} with random weights θ, θ^-
2: Initialize replay buffer $\mathcal{D}$ with capacity N
3: Set initial battery charge $B \leftarrow 80$, ϵ, γ, α, and decay rates
4: **for** $k = 1$ to K **do**
5: $\quad s \leftarrow S$, $battery \leftarrow B$, $path \leftarrow [s]$, $done \leftarrow$ **False**
6: $\quad$ **while not** *done* **do**
7: $\quad\quad$ Select a using ϵ-greedy:
With probability ϵ, select random a
Otherwise, $a \leftarrow \arg\max_{a'} Q_\theta(s, battery, a')$
8: $\quad\quad$ Execute a, observe r, next state s', next battery $battery'$
9: $\quad\quad$ Store $(s, a, r, s', done, battery, battery')$ in $\mathcal{D}$
10: $\quad\quad$ $s \leftarrow s'$, $battery \leftarrow battery'$, Append s to $path$
11: $\quad\quad$ **if** replay buffer $\mathcal{D}$ has enough samples **then**
12: $\quad\quad\quad$ Sample minibatch from $\mathcal{D}$
13: $\quad\quad\quad$ **for** each transition $(s_i, a_i, r_i, s'_i, done_i, battery_i, battery'_i)$ **do**
14: $\quad\quad\quad\quad$ $y_i = r_i + \gamma \max_{a'} Q_{\theta^-}(s'_i, battery'_i, a')(1 - done_i)$
15: $\quad\quad\quad$ **end for**
16: $\quad\quad\quad$ Compute loss: $L = \frac{1}{|\text{batch}|} \sum_i (y_i - Q_\theta(s_i, battery_i, a_i))^2$
17: $\quad\quad\quad$ Perform gradient descent on L to update θ
18: $\quad\quad$ **end if**
19: $\quad\quad$ **if** every C steps **then**
20: $\quad\quad\quad$ Update target network: $\theta^- \leftarrow \theta$
21: $\quad\quad$ **end if**
22: $\quad\quad$ **if** $s = D$ **or** $battery \leq 0$ **then**
23: $\quad\quad\quad$ $done \leftarrow$ **True**
24: $\quad\quad\quad$ **if** current_reward $>$ best_reward **then**
25: $\quad\quad\quad\quad$ Update best epoch results with current path and metrics
26: $\quad\quad\quad$ **end if**
27: $\quad\quad$ **end if**
28: $\quad$ **end while**
29: $\quad$ Record best total reward, distance, travel time
30: $\quad$ Decay ϵ towards its minimum value
31: **end for**
32: **return** Best epoch results containing optimal path

Finally, the algorithm checks for episode termination conditions within the while-loop. If the destination is reached or the battery is exhausted, the episode is terminated (line

25). Moreover, if the current reward surpasses the best recorded reward, the algorithm updates its best epoch results with the current path and associated metrics (lines 27-29). At the end of each episode, the best total reward, distance, and travel time are recorded, and the exploration rate ϵ is decayed (lines 29-30). Ultimately, after iterating through all episodes, the algorithm returns the best epoch results, which include the optimal path and performance metrics, thereby effectively addressing the constrained path planning problem through deep Q-learning (line 31).

4.5 Performance Evaluation

To achieve better results compared to previous work [272, 273] for large graphs, in this section, we compare the performance of the implemented methods: TQL, Q-Learning, and DQL.

Our evaluation centers on dynamic environments, where we measure performance in terms of overall travel time, total distance traveled, accumulated rewards, and changes in Q-values. To understand the impact on performance, we vary key parameters such as the graph size, the number of charging stations, and the distance between the source and destination. Furthermore, to account for battery life in our proposed methods, we conducted empirical tests using a maximum battery level of $B_{max} = 80\%$ and a minimum battery level of $B_{min} = 20\%$. Additionally, the EV begins its journey with an initial charge of 80%.

In our implementation for all the algorithms, we used exponential decay for learning rate (α) and exploration rate (ϵ) for $\epsilon-$greedy approach. Exponential decay can be represented as follows:

$$\theta_{t+1} = max(\theta_{min}, \theta_t \times decay_rate) \tag{4.4}$$

where θ can be the learning rate (α) or exploration rate (ϵ), θ_t is the parameter value at epoch t. $decay_rate$ is the constant multiplier set to (0.999), and θ_{min} is the lower bound to prevent the parameter from decaying indefinitely. We started with $\alpha = 0.1$, and $\epsilon = 0.25$. The lower bound (minimum) for α and ϵ is 0.001 to ensure the agent does not explore after an adequate number of explorations at first. Other key parameters in our implementation are the discount factor γ, which is set to 0.9, to encourage the agent to value long-term reward as the goal is to minimize the travel time from the source to the destination (considering the longevity of the battery as well). Also, the number of episodes K is set to be equal to 2000 to ensure sufficient training time to achieve the best result.

Figure 4.2: Comparing the traveled distance (in KM) of different methods (TQL, Q-Learning, and DQL) for (a) 125 nodes, (b) 250 nodes, and (c) 500 nodes.

4.5.1 Graph Generation & Charging Station Placement

To evaluate the performance of our proposed methods, we generate graphs based on real-world maps, varying in size and comprising nodes connected by weighted edges. To create graphs with a specific number of nodes, we extract a subset from a larger generated graph that meets the desired node count. Given the current range of EV batteries, an electric vehicle can typically travel from any starting point to a destination within a city without requiring a recharge. However, for our experiments, we extend the distances between edges to evaluate how effectively our algorithms can identify charging stations along the route.

In addition, we attribute three unique pairs of traffic speed and battery consumption

values to each edge based on its length. This configuration enables us to mimic varying traffic conditions and dynamically adjust the EV's speed and battery usage at every time step. We also designate specific nodes as charging stations, ensuring that every node is within reach of at least one charging station, even under extreme conditions of battery consumption. This approach guarantees that each algorithm we test always has at least one viable solution.

(a) (b) (c)

Figure 4.3: Comparing the traveled time (in Minutes) of different methods (TQL, Q-Learning, and DQL): (a) 125 nodes, (b): 250 nodes, and (c): 500 nodes

Our method for positioning charging stations is systematic. We start by randomly choosing a node in the graph to act as the initial charging station and add it to the list I_{Station}. For every subsequent charging station, we select the node that is the furthest away in terms of battery consumption from the most recently chosen station and that can be reached with the maximum battery charge B_{Max}. This recursive process continues until

Figure 4.4: Comparing the achieved cumulative reward of different methods (TQL, Q-Learning, and DQL): (a) 125 nodes, (b): 250 nodes, and (c): 500 nodes

the entire graph is covered, ensuring that every area is accessible without depleting the battery.

Finally, we randomly pick a source and a destination node within a predetermined distance range for the EV's journey. For simplicity, we classify each road into one of three traffic conditions, which enables us to simulate and assess the impact of varying traffic statuses on the EV's trip.

4.5.2 Performance Evaluation over Different Networks

To evaluate the proposed methods, we tested them across three different network configurations, each containing 125, 250, or 500 nodes. These variations allowed us to assess performance across diverse network scales.

For each network, we report four main metrics. Traveled distance (in kilometers) and time (in minutes), cumulative reward that the agent received over all the episodes, and Q-values changes. Analyzing these metrics will give us a detailed perspective of the functionality of each algorithm, the effectiveness of each, and also the convergence. Then, for the network with $N = 125$, we will examine the effects of changing the placement and the number of charging stations on two main metrics, traveled distance and time.

Traveled Distance: Fig. 4.2 presents a detailed comparison of travel time across three different network configurations. As shown, DQL consistently outperforms the other algorithms and converges faster. However, in the case of 500 nodes, DQL initially struggles in the early episodes, unlike the smaller networks, where it quickly finds the optimal solution. Q-Learning, on the other hand, only surpasses TQL in the $N = 125$ scenario, while in larger networks, it struggles to converge until the final episodes. TQL generally performs as well as or better than Q-Learning, benefiting from its initialization with the heuristic-based Dynamic TERC2 results. In the $N = 125$ case, TQL exhibits noisy behavior due to over-exploitation of suboptimal results from Dynamic TERC2. However, as it explores alternative actions, it achieves varied but non-optimal outcomes.

The same trend can be seen in Fig. 4.3 as traveling time and distance have a linear correlation with each other. In our case, travel time between each node is a linear combination of the distance, which is calculated as follows:

$$distance = G[N][N+1] \tag{4.5}$$

where N and N+1 are the adjacent nodes. Travel time is then calculated based on two factors that were derived from maps that were used in the real world: base speed and traffic factor.

$$speed = \frac{base_speed}{traffic_factor} \tag{4.6}$$

$$travel_time = \frac{distance}{speed} \tag{4.7}$$

Cumulative Reward: Fig. 4.4 illustrates the cumulative reward for each method. DQL consistently achieved fewer negative rewards across all networks, ultimately attaining the optimal negative cumulative reward. Similar to travel time and distance, TQL outperformed Q-Learning in networks with 250 and 500 nodes. It is important to note that the y-axis in Fig. 4.4(c) is scaled by 10^6.

As with travel time and distance, DQL converged rapidly, making it the most effective approach for this problem, as minimizing negative rewards is one of the key objectives.

Furthermore, all methods successfully identified the best path for the given instances and converged to the optimal policy. However, DQL reached the optimal policy (π^*) faster than TQL, while Q-Learning demonstrated the weakest performance.

Altering the Number of Charging Stations: As discussed earlier, the location and number of charging stations follow a recursive systematic order. To determine whether to place a charging station at a given node, we analyzed the effect of the accessibility radius on travel distance. Increasing this radius results in fewer charging stations across the map. Fig. 4.5 illustrates these effects. For instance, with a radius of 60, resulting in a sparse distribution of charging stations, all methods required more time to find the optimal policy due to charging constraints. This finding suggests that maintaining an adequate number of charging stations is crucial in real-world scenarios. To ensure a more realistic setup aligned with real-world conditions, we set the baseline number of charging stations using a radius of 60.

4.6 Conclusion

This study explores the potential of advanced RL techniques to optimize EV routing and charging in dynamic urban environments. Our comparative analysis demonstrates that while heuristic-enhanced Q-Learning accelerates convergence and improves initial performance over traditional Q-Learning, DQL exhibits superior robustness and efficiency in real-time decision-making. DQL consistently achieves lower travel times, reduced energy consumption, and improved overall performance by effectively balancing route optimization and battery management under varying traffic and environmental conditions.

However, it is important to note that both the RL methods and heuristic approaches have limitations. For instance, the performance of deep RL is highly sensitive to hyperparameter selection and the quality of training data, which may hinder rapid adaptation in highly variable real-world scenarios. Additionally, while heuristic initialization (as used

in TQL) can speed up convergence, it may also introduce biases that lead to suboptimal policies if the underlying heuristic does not adequately capture the dynamics of all environmental conditions.

These findings underscore the promise of DRL for smart transportation and its relevance in designing future EV systems that are both sustainable and adaptable. Future research should focus on integrating heterogeneous data sources and adaptive learning mechanisms to bridge the gap between simulation and large-scale practical deployments, while also addressing the scalability and stability challenges identified in this work.

Figure 4.5: Comparing the traveled distance of different methods (TQL, Q-Learning, and DQL) in a network of 125 nodes: (a) radius = 30, (b) radius = 45, and (c) radius = 60 (baseline).

Chapter 5

Vehicle Patrol Scheduling

This chapter examines the Vehicle Patrol Scheduling (VPS) problem, focusing on how to plan efficient patrol routes under constraints such as shift durations, mandatory rest periods, maximum revisit intervals, and dynamic emergency responses. We begin by reviewing and implementing a classical optimization model, a meta-heuristic genetic algorithm approach (GDVPS) the Adaptive Hill-Climbing-Based Patrol Scheduling (AHBPS) heuristic, both of which provide immediate adaptability to unforeseen changes. We then introduce a learning-based approach called $\mathcal{P}^4\mathcal{O}$ (Patrol Planning with Proximal Policy Optimization), which leverages Reinforcement Learning to improve route planning through accumulated experience. By integrating these complementary perspectives, the chapter presents a comprehensive framework for dynamic and scalable VPS that balances short-term responsiveness with long-term efficiency.

5.1 Introduction

VPS is a fundamental problem in urban safety management, aiming to plan efficient vehicle routes that ensure both wide-area coverage and timely responses to emerging incidents. VPS belongs to the broader class of Vehicle Routing Problems (VRP), initially introduced by Dantzig et al. [282], but it incorporates additional operational constraints such as shift durations, mandatory rest periods, maximum revisit intervals, and dynamic emergency responses. These constraints, combined with the inherent NP-hard nature of VPS, make it significantly more challenging than classical VRP formulations.

In prior studies, VPS has been addressed from two primary perspectives: optimization and heuristic techniques for immediate operational efficiency and learning-based ap-

proaches for adaptive policy development. One line of research formalized the VPS problem as a mathematical optimization model, capable of providing optimal solutions in small, static instances. To address larger and more dynamic environments, heuristic methods such as Adaptive Hill-Climbing-Based Patrol Scheduling (AHBPS) were introduced. AHBPS employs an iterative local search approach to refine vehicle routes and incorporates mechanisms for real-time adjustment to unforeseen changes, such as emergent emergency events or dynamic traffic conditions.

While heuristic methods offer computational efficiency and adaptability, they do not explicitly learn from historical patrol patterns or optimize for long-term performance across multiple shifts and incidents. To address these limitations, a second line of research framed VPS as a Markov Decision Process (MDP), enabling the application of Reinforcement Learning (RL) techniques. Specifically, the Patrol Planning with Proximal Policy Optimization ($\mathcal{P}^4\mathcal{O}$) approach was proposed, utilizing the Proximal Policy Optimization (PPO) algorithm to learn near-optimal patrol policies through repeated interactions with a simulated urban environment. $\mathcal{P}^4\mathcal{O}$ demonstrated strong performance in balancing routine patrol coverage with responsive emergency handling, particularly in complex metropolitan networks. Both approaches, heuristic optimization and reinforcement learning, highlight the fundamental trade-off in VPS between short-term adaptability and long-term optimality. Extensive evaluations conducted on real-world urban maps revealed that heuristic methods like AHBPS can respond quickly to local changes, while learning-based methods like $\mathcal{P}^4\mathcal{O}$ leverage accumulated experience to develop more globally efficient policies over time. This complementary nature suggests that effective patrol scheduling for urban safety requires integrating real-time optimization techniques with learning-based policy refinement.

This chapter integrates insights from both optimization and learning-based approaches, laying the foundation for a comprehensive framework for dynamic and scalable vehicle patrol scheduling. By combining the immediate adaptability of heuristic methods with the experience-driven policy improvement of reinforcement learning, the proposed framework aims to enhance both the coverage and responsiveness of vehicle patrol operations in urban environments.

Over the next subsection, we analyze some relevant studies in the literature that employed heuristic, optimization, and RL approaches.

5.2 Related Work

VPS is a specialized case of VRP, which has been extensively studied in logistics, operations research, and Intelligent Transportation Systems [283]. Our review is structured into two main subsections: the first examines research employing optimization and heuristic methods to tackle problems related to VPS and VRP, while the second explores studies that integrate RL as a core methodology.

5.2.1 Optimization and Heuristic-Based Approaches

Several studies have explored VPS and VRP through optimization and heuristic techniques. The research in [284] addressed the challenge of organizing security teams to patrol a mass rapid transit network, with an emphasis on optimizing patrol assignments while introducing randomness to enhance unpredictability. Their mathematical programming model was tested on the Singapore Mass Rapid Transit system, showing improved security results by making patrol schedules less predictable.

In a similar vein, [285] proposed a mathematical programming method to enhance police patrol scheduling, intending to balance limited resources with high demand in crime-prone areas. The study presented two models, independent and integrated planning, both designed to minimize patrol time while maximizing coverage. Using real data from a local police department, the integrated model was found to be more effective than the independent one, leading to better resource allocation and shorter patrol durations.

A different perspective was explored in [286], where the emphasis was on improving the deployment of traffic police patrols to boost law enforcement effectiveness. The authors developed a Multi-Objective Linear Programming model with three main goals: maximizing police presence, focusing on high-risk areas, and optimizing the timing for proactive enforcement.

In the realm of maritime security, the scheduling of patrol boats was examined in [287]. This study employed a column generation method within a branch-and-price framework to identify the minimum number of patrol vessels needed for ongoing maritime surveillance. A subsequent study [288] enhanced this approach by factoring in resource constraints like refueling and replenishment times, ensuring consistent patrol coverage.

In vehicle routing, [289] introduced a set-based Particle Swarm Optimization (PSO) algorithm specifically for VRP with Time Windows (VRPTW). Their proposed Set-based PSO (S-PSO-VRPTW) utilized a time-oriented nearest neighbor heuristic for updating positions and included a novel decision-making function to balance both primary and

secondary objectives. Computational experiments showed that this method outperformed existing heuristics and metaheuristics, especially in scenarios with large time windows and vehicle capacities.

Additionally, the Vehicle Routing Problem with Simultaneous Delivery and Pickup and Time Windows (VRPSDPTW) was examined by [290]. The authors developed a multi-objective optimization model that included five criteria: minimizing the number of vehicles, total travel distance, makespan, total waiting time, and total delay time. They proposed two algorithms, Multiobjective Local Search (MOLS) and Multiobjective Memetic Algorithm (MOMA), demonstrating that MOLS consistently outperformed MOMA across benchmark instances by achieving superior convergence and diversity in Pareto front approximations.

The challenge of dynamic vehicle routing in the face of traffic congestion was examined in [291]. The authors put forward a Markov Decision Process (MDP) model that took into account both time-dependent and stochastic travel times, addressing the shortcomings of conventional routing methods. By employing a rollout algorithm grounded in approximate dynamic programming, they successfully alleviated the curse of dimensionality. Their experimental findings, which utilized logistics data from Singapore, revealed substantial cost savings compared to static routing methods, underscoring the necessity of incorporating real-world traffic dynamics into VRP optimization.

To tackle the challenges of large-scale city patrols, [292] proposed a decomposition and grafting strategy for city-scale patrolling (CSP). Their approach frames CSP as an integer programming (IP) problem focused on maximizing the police visibility rate (PVR) while ensuring timely responses to emergency situations. By breaking the problem down into two loosely connected subproblems, minimizing police deployment (MinP) and maximizing PVR (MaxP), the authors introduced efficient polynomial-time algorithms for each. A grafting mechanism was also developed to combine the solutions of these subproblems, significantly improving the scalability and effectiveness of city-wide patrol planning. This method was validated with real-world data from China, demonstrating enhanced performance in managing detailed patrol schedules across extensive urban areas.

For UAV-based patrol operations, [293] proposed a model for task assignment that takes into account multiple bases and rechargeable endurance. This model enhances traditional vehicle routing by incorporating UAV-specific limitations, such as restricted endurance, various recharging locations, and different types of UAVs. They developed a time-priority immune clonal selection algorithm that initially creates an optimal sequence for task assignments and then refines UAV paths using a time-priority approach. Experimental tests involving up to 100 task points demonstrated that their method significantly cuts down completion time compared to existing UAV task assignment strategies, making it particu-

larly effective for extensive aerial patrol missions.

The authors in [284] addressed the challenge of assigning security teams to patrol a mass rapid transit network. The objective was to optimize the scheduling of security teams while considering limited resources, rostering, and security requirements. A mathematical programming model was proposed to solve the problem by incorporating randomness into patrol timings, making it more difficult for adversaries to predict patrol schedules. The model was tested on the Singapore mass rapid transit system, demonstrating how randomized patrol schedules can enhance security by reducing vulnerability to threats.

A mathematical programming approach for optimizing police patrol scheduling was presented in [285]. This study focused on improving the efficiency of patrol duties in the context of an imbalance between limited police resources and the high demands of crime-prone areas. Two models were proposed: independent planning and integrated planning, both aiming to minimize patrol time while maximizing service coverage. The models were tested using real data from a local police department, with the integrated model demonstrating better results by reducing patrol time and improving resource allocation.

In [294], researchers addressed the problem of constructing an optimal patrol boat schedule to fully cover a maritime area. They proposed applying a column generation approach within a branch-and-price framework to determine the minimum number of boats needed for continuous patrols, taking replenishment breaks into account. The methodology was demonstrated on a sample network, and the results provided valuable insights into the trade-offs between boat endurance, replenishment time, and fleet size.

In [295], the authors presented a patrol route optimization study for incident response vehicles aimed at ensuring timely responses to traffic incidents, improving safety, and reducing congestion. They proposed a mixed-integer nonlinear programming model to minimize patrolling travel costs while considering the expected maximum response time from any location to all incident hotspots. To solve this route optimization problem, a Column Generation-based solution technique was introduced. Additionally, a unified framework combining a genetic algorithm with the column generation approach was applied to assess the impact of dispatch station locations on routing costs.

The authors in [286] focused on optimizing the deployment of traffic police patrol vehicles to maximize the benefits for road users, enhancing road safety and law enforcement effectiveness. To achieve this, they proposed a Multi-Objective Linear Programming (MOLP) model with three key objectives: maximizing police presence and visibility, prioritizing patrols at locations prone to traffic offenses, and optimizing time for proactive enforcement.

The scheduling of security teams for patrolling a Mass Rapid Transit (MRT) rail net-

work in a large urban city was studied in [296]. The objective was to assign security teams to patrol different stations at varying times, subject to security constraints. The patrol scheduling problem was defined as an assignment problem, where security teams are assigned to patrol stations across different time periods. The goal was to maximize coverage while minimizing the predictability of the patrol schedule for security purposes.

In [297], the optimization of bus schedules was addressed using an improved multi-objective genetic algorithm. This refined genetic algorithm aimed to tackle the complexities of bus scheduling by focusing on minimizing both operational costs and passenger wait times. The algorithm enhanced traditional genetic approaches by incorporating multi-objective optimization techniques, leading to a more efficient and effective scheduling process.

An optimization approach to the problem, known as the Patrol Boat Scheduling Problem with Complete Coverage, was investigated in [287]. The study employed a column generation approach within a branch-and-price framework to determine the minimum number of patrol vessels needed to ensure continuous coverage of maritime environments. In a subsequent study [288], the same problem was addressed using optimization techniques, focusing on resource limitations such as refueling and replenishment times. The authors provided an efficient scheduling solution within the context of naval operations, enabling continuous patrols in predefined regions while enhancing maritime security.

In [298], the scheduling of highway patrol drones was studied using both mathematical programming and a heuristic algorithm to optimize patrol routes and meet varying patrol demands. Two integer programming models were proposed; however, due to computational limitations, a column generation-based heuristic was introduced to improve efficiency. Tested on real-world highways in Korea, the heuristic demonstrated promising results in enhancing safety and reducing operational costs.

Upon reviewing the literature, it is evident that a gap exists in the domain of VPS. This study aims to address this gap by introducing dynamic, robust, and scalable approaches to tackle the real-world challenges faced by security companies, distinguishing our work from conventional methods in existing research.

5.2.2 RL-Based Approaches

Recent advances in RL have led to innovative solutions for challenges in patrol and vehicle routing. In [299], the authors utilized Deep RL to optimize UAV patrol routes for urgent rescue operations in areas affected by disasters. Their research showcased how Deep Q-Learning, along with its enhanced version, Double Deep Q-Learning, allowed UAVs to adjust their routes dynamically in response to environmental constraints and mission goals.

Similarly, [300] integrated multi-agent Geo-simulation with RL to improve UAV patrolling strategies in ever-changing environments. Their method enabled multiple UAVs to work together in tracking moving targets while avoiding obstacles, using Geo-simulation to create realistic settings and RL for flexible decision-making.

The challenge of dynamic vehicle routing in the face of traffic congestion was examined in [291]. The authors introduced an MDP model that took into account both time-dependent and stochastic travel times, addressing the shortcomings of conventional routing methods. By employing a rollout algorithm grounded in approximate dynamic programming, they successfully alleviated the curse of dimensionality. Their experimental findings, which utilized logistics data from Singapore, revealed substantial cost savings compared to static routing methods, underscoring the necessity of incorporating real-world traffic dynamics into VRP optimization.

The coordination of police patrols using MARL was explored in [301]. This research focused on the dynamic rescheduling of patrols in response to real-time incidents, aiming to strike a balance between maintaining a visible patrol presence and addressing emerging threats. By integrating Multi-Agent Value Function Approximation with a decentralized optimization heuristic, the authors developed a method for effectively learning dispatch and rescheduling policies.

Further developments in RL-driven patrol scheduling were discussed in [302], where the authors introduced a hybrid RL-heuristic approach for real-time police patrol rescheduling. This method tackled the challenge of balancing proactive patrol strategies with incident response, showcasing the effectiveness of RL in addressing dynamic, bi-objective optimization problems.

A new application of Asynchronous PPO for dynamic police patrol scheduling was presented in [303]. This study addressed the challenge of balancing police presence with incident response by utilizing RL to develop dispatch rules, enabling real-time adjustments to patrol schedules as incidents arise.

While the studies mentioned have utilized RL for patrol scheduling across various fields, including UAV surveillance and police patrol coordination, they mainly concentrate on either static patrol planning or reactive scheduling. In contrast, our research presents a new RL framework for VPS that functions in a dynamic, real-time setting. Unlike earlier methods that rely on fixed patrol locations or use multi-agent reinforcement learning (MARL) solely for coordination, we approach the problem as an MDP and use PPO to dynamically enhance patrol strategies. Furthermore, we incorporate an adaptive emergency handling system, enabling the agent to modify vehicle routes in real time while ensuring optimal patrol coverage.

5.3 Optimization, Heuristic, and Genetic Approaches

In this section, we solve the VPS problem with three different approaches: optimization model that is suitable for smaller environments, a hill-climbing heuristic approach, which is designed to work efficiently on both large and small scale environments in a short amount of time, and lastly, a genetic algorithm-based approach, which yields near-optimal solutions. We first start by introducing the system model and problem definition.

5.3.1 System Model and Problem Definition

In this section, we present the system model utilized in this study, describing its structure and key components in detail. We also provide a concise overview of the problem definition, laying the groundwork for the subsequent analysis and solution approaches.

System Model

As illustrated in Fig. 5.14, we conceptualize the operational environment for a fleet of patrol vehicles as a weighted directed graph $G = (N, E)$, where N represents the set of nodes, which includes a depot $l \in N = 0$, and a set of locations $l = \{1, 2, ..., |N|\}$ that must be patrolled regularly. Each location has a minimum patrolling time T^{patrol} and a maximum allowable gap between visits T^{gap}. The set E signifies the links representing possible routes between these nodes, with weights corresponding to travel time between locations, denoted as $W_{l,l'}$, which are measured based on current traffic conditions. These weights are dynamically updated to reflect the real-world nature of the problem. During the operational period, vehicles in the fleet ($c \in C$) are systematically assigned routes within the graph. Each route begins at the depot, visits designated locations, and returns to the depot upon completing its shift. Each vehicle must adhere to a mandatory rest time T^{rest} after a maximum operational duration of T^{shift}. This structure ensures comprehensive coverage of all areas of interest during the operational time. The objective of this study is to optimize the routing of vehicles in a network to maximize the total visits to all patrolled locations for security within the designated operational period. This involves planning routes from the depot to various locations and back while ensuring the mandatory rest is taken at the depot.

Figure 5.1: Illustration of the system model: Locations are numbered and distinguished by different colors. Location l_2 has not been patrolled by either of the vehicles (Red cars labeled c_1 & c_2). Vehicle c_2 is en route to patrol location l_3, while location l_4 has been securely patrolled and must be revisited within a specified time gap. The dotted lines represent the routes taken by each vehicle, with different colors indicating different routes or purposes. For example, vehicle c_2 starts at the depot, visits location l_4, and is traveling to location l_3. These routes are marked with black dots. The white lines represent the path back to the depot, while dark blue lines indicate the route for revisiting locations. This process occurs within a single shift, and in subsequent shifts, different or similar routes may be taken. Vehicle c_2's routes are depicted with varying colors to represent different scenarios. The depot, indexed by D, serves as the starting and ending point for all vehicles.

Problem Definition

As stated earlier, our primary goal is to increase the total number of locations visited throughout the workday. To achieve this, we ensure no more than one vehicle visits the same location simultaneously and each location is revisited within a specified gap to maintain continuous safety.

Problem Definition (maximizing the number of visited locations): *Given a graph $G = (N, E)$, where N and E respectively represent the set of nodes and links, let there be a set of locations $l \in N = 1, 2, ..., |N|$. The problem is to determine the optimal route for each vehicle $c \in C$ during each shift, starting from the depot l=0, while visiting the maximum number of locations and returning to the depot. This must be done while*

considering the maximum operation time for each shift T^{shift}, rest time T^{rest}, minimum location patrolling time T^{patrol}, and the maximum allowable gap between visits T^{gap}. The objective is to maximize the total number of locations visited during the entire operational period.

Proof of NP-Hardness

In this subsection, we prove that VPS is NP-hard by reducing the Traveling Salesman Problem (TSP) to it.

Theorem 1. *The Vehicle Patrol Scheduling problem (VPS) is NP-hard.*

Proof. We prove the VPS problem is NP-hard by reducing the well-known NP-hard problem, TSP [304], to VPS. Let $G = (N, E)$ be the graph corresponding to an instance of TSP, where N is the set of cities and E is the set of edges between the cities, with each edge $(l, l^{'}) \in E$ representing the distance $d_{ll'}$ between city l and city $l^{'}$. The objective in TSP is to find a Hamiltonian cycle of minimum length that visits each city exactly once and returns to the starting city. We construct a corresponding graph $G' = (P, L)$ for the VPS problem as follows:

- Let the set of patrol locations P in G' corresponds to the set of cities N in G. Thus, $P = N$.
- Let the set of edges L in G' corresponds to the edges E in G, with identical distances. Hence, $L = E$ and $d_{ll'} = d_{ll'}$ for each $(l, l^{'}) \in L$.
- Add a depot node D to G', and connect it to every node in P with arbitrary positive weights, say $t_{D,l} = 1$ for each $l \in P$. This represents the starting and ending points of the patrol vehicle.

Next, we show that solving VPS on this instance is equivalent to solving TSP:

1. Set the number of vehicles $C = 1$.
2. Assign a length (or travel time) $t_{ll'}$ to each edge $(l, l^{'}) \in L$ in G', where $t_{ll'} = d_{ll'}$, the same as the distances between cities in the TSP instance.
3. Set the patrol times at the locations to be zero, i.e., no time is spent at any node apart from traveling.

4. The objective in VPS is to maximize the number of distinct locations visited. Since there is only one vehicle and the patrol times are zero, the vehicle must visit every location exactly once and return to the depot in a cycle.

Now, consider TSP. If an optimal solution to TSP exists, it will visit each city exactly once and return to the starting city, minimizing the total travel distance. This optimal tour corresponds to a solution of the VPS problem where the vehicle departs from the depot, visits each location exactly once, and returns to the depot. The total number of locations visited in VPS corresponds to the number of cities visited in TSP. Thus, an optimal solution to TSP corresponds to an optimal solution to VPS, and solving the VPS problem would also solve TSP. Since TSP is known to be NP-hard, and we can reduce any instance of TSP to an instance of VPS in polynomial time, it proves that the VPS problem is also NP-hard.

□

5.3.2 Problem Formulation

In this section, we mathematically formulate the problem to derive the optimal solutions. The notations used are listed in Table 5.1. Let V_l denote the total number of visits to location l. The objective of the problem is to maximize this number across all locations, which can be mathematically expressed as:

$$\text{Maximize} \quad \sum_{l=1}^{|N|} V_l \tag{5.1}$$

Subject to: (5.2)-(5.16), where these constraints are introduced in detail below.

Initial Location Constraint: Each vehicle c should start its journey from the depot (D) as the initial starting point. If we let $X_{c,l,v}$ to indicate whether vehicle c has done its v-th visit at location l, then the initial starting location for each vehicle can be written as:

$$X_{c,D,1} = 1 \qquad \forall c \in C \tag{5.2}$$

Operation shift Constraint: If we let $Y_{c,l,v,s}$ indicate that vehicle c has done its v-th visit at location l at s-th shift, then the following constraint ensures that if vehicle c has not done its v-th visit at location l, then this visit did not take place during any shift. That is, if $X_{c,l,v}$ is zero, so is $Y_{c,l,v,s}$.

$$Y_{c,l,v,s} \leq X_{c,l,v} \qquad \forall c \in C, l \in N, v = 1..\boldsymbol{v}^{\max}, s = 1..\boldsymbol{s}^{\max} \tag{5.3}$$

Table 5.1: Notations used for the problem formulation

Decision Variables	
$V_l \geq 0$	Total number of visits to location l.
$X_{c,l,v} \in \{0,1\}$	Indicates whether vehicle c has done its v-th visit at location l.
$Y_{c,l,v,s} \in \{0,1\}$	Indicates whether vehicle c has done its v-th visit at location l at s-th shift.
$A_{c,v} \geq 0$	Time when vehicle c starts its v-th visit.
$P_{c,v} \geq 0$	Patrolling time of vehicle c on its v-th visit.
$L_{l,s} \geq 0$	Time when location l has been visited on s-th shift.
Parameters	
T^{shift}	Maximum operation time at each shift.
$W_{l,l'}$	Travel Time between locations l and l'.
$Adj_{l,l'}$	Indicating the adjacency of locations l and l'.
T^{rest}	Minimum rest time of a vehicle.
D	The depot location index.
T^{patrol}	Minimum patrolling time of a location.
T^{gap}	Maximum gap between consecutive visits.
R_c^{start}	Start of the rest time of vehicle c.
R_c^{end}	End of the rest time of vehicle c.

Single Vehicle Constraint: As enforced by the following constraint, a vehicle can only visit one location at a time.

$$\sum_{l \in N} X_{c,l,v} = 1 \qquad \forall c \in C, v = 1..\boldsymbol{v}^{\max} \tag{5.4}$$

Single Location Constraint: No location can be patrolled by multiple vehicles at the same time as enforced by the following:

$$\sum_{c \in C} \sum_{v=1}^{\boldsymbol{v}^{\max}} Y_{c,l,v,s} \leq 1 \qquad \forall l \in N, s = 1..\boldsymbol{s}^{\max} \tag{5.5}$$

Path Feasibility Constraint: A vehicle cannot travel between two locations unless a path exists. This constraint ensures that the vehicle's movements are feasible within the

road network.

$$X_{c,l,v} \leq \sum_{l' \in N} X_{c,l',v-1} \cdot Adj_{l,l'} \forall c \in C, v = 1..\boldsymbol{v}^{\max}, l \& l' \in N \tag{5.6}$$

Depot Visit Constraint: A vehicle should visit the depot at least once during each shift. This ensures that vehicles return to the depot for maintenance, refueling, rest, etc.

$$\sum_{v=1}^{\boldsymbol{v}^{\max}} \sum_{s=1}^{\boldsymbol{s}^{\max}} Y_{c,\mathrm{D},v,s} \geq 1 \quad \forall c \in C \tag{5.7}$$

Depot Rest Time Constraint: A vehicle must stay at the depot for a duration of T^{rest} to ensure both vehicles and personnel receive adequate rest before resuming to patrol. Let $P_{c,v}$ be the patrolling time of vehicle c on its v-th visit, then it must be at least T^{rest} when the vehicle is at the depot.

$$P_{c,v} \geq T^{rest} \cdot \sum_{s=1}^{\boldsymbol{s}^{\max}} Y_{c,D,v,s} \quad \forall c \in C, v = 1..\boldsymbol{v}^{\max} \tag{5.8}$$

Vehicle Visit Time Constraint*: In the following, we calculate the time when vehicle c starts its v-th visit ($A_{c,v}$), which can be obtained by summing the time when the vehicle started its previous visits ($A_{c,v-1}$), patrolling time ($P_{c,v-1}$), and the time needed to travel from location l to l', denoted as $W_{l,l'}$.

$$A_{c,v} = A_{c,v-1} + P_{c,v-1} + \sum_{l \in N} \sum_{l' \in N} W_{l,l'} \cdot X_{c,l,v-1} \cdot X_{c,l',v} \quad \forall c \in C, v = 1..\boldsymbol{v}^{\max} \tag{5.9}$$

Rest Time End Constraint*: The time when vehicle c finishes its rest, denoted as R_c^{end}, must occur after the vehicle enters the depot and completes its required rest period.

$$\sum_{s=1}^{\boldsymbol{s}^{\max}} Y_{c,D,v,s} \cdot (A_{c,v} + P_{c,v}) \leq R_c^{end} \quad \forall c \in C, v = 1..\boldsymbol{v}^{\max} \tag{5.10}$$

Rest Time Start Constraint: The rest time for a vehicle, denoted by R_c^{start}, begins when it arrives at the depot.

$$A_{c,v} \geq R_c^{start} \cdot \sum_{s=1}^{\boldsymbol{s}^{\max}} Y_{c,D,v,s} \quad \forall c \in C, v = 1..\boldsymbol{v}^{\max} \tag{5.11}$$

Maximum Shifts Constraint: A location cannot be visited and patrolled more than the number of possible shifts.

$$\sum_{c \in C} \sum_{v=1}^{\boldsymbol{v}^{\max}} Y_{c,l,v,s} = (s < V_l) \qquad \forall l \in N, s = 1..\boldsymbol{s}^{\max} \tag{5.12}$$

where $(s < V_l)$ is a Boolean expression that evaluates to 1 if the condition is true, and 0 otherwise.

Vehicle Arrival Constraint: Constraint 5.13 ensures that the patrolling of location l by vehicle c during the s-th shift cannot start until the vehicle has arrived at the location. Here, $L_{l,s}$ represents the time when location l was visited during the s-th shift.

$$Y_{c,l,v,s} \cdot A_{c,v} \leq L_{l,s} \quad \forall c \in C, l \in N, v = 1..\boldsymbol{v}^{\max}, s = 1..\boldsymbol{s}^{\max} \tag{5.13}$$

Consecutive Visits Gap Constraint*: This constraint ensures that a location cannot remain unpatrolled for longer than a duration of T^{gap}, where T^{gap} represents the maximum allowable gap between consecutive visits. In particular, the arrival time of vehicle c to location l during its v-th visit in the s-th shift, when $Y_{c,l,v,s} = 1$, must occur after the start time of the $(s-1)$-th shift (i.e., $L_{l,(s-1)}$), plus the patrol time T^{patrol} and gap time T^{gap}).

$$\begin{aligned} A_{c,v} \geq Y_{c,l,v,s} \cdot (L_{l,(s-1)} + T^{gap} + T^{patrol}) \\ \forall c \in C, l \in N, v = 1..\boldsymbol{v}^{\max}, s = 2..\boldsymbol{s}^{\max} \end{aligned} \tag{5.14}$$

Minimum Patrolling Duration Constraint*: Each location visited must be patrolled for a minimum duration of T^{patrol} to ensure sufficient security. In the following constraint, the arrival time of vehicle c plus the patrolling duration must occur after the start of patrolling and the minimum allowable patrol time T^{patrol}.

$$\begin{aligned} A_{c,v} + P_{c,v} \geq (T^{patrol} + L_{l,s}) \cdot Y_{c,l,v,s} \\ \forall c \in C, l \in N, v = 1..\boldsymbol{v}^{\max}, s = 1..\boldsymbol{s}^{\max} \end{aligned} \tag{5.15}$$

Shifts Order Constraint: The patrolling shifts must occur in order, and the difference between them must exceed the minimum patrol time (T^{patrol}) plus the gap time (T^{gap}).

$$\begin{aligned} L_{l,s} \geq L_{l,s-1} + (s < V_l) \cdot (T^{patrol} + T^{gap}) \\ \forall l \in N, s = 1..\boldsymbol{s}^{\max} \end{aligned} \tag{5.16}$$

Constraints marked with "*" are not linearized. Their linearized forms are provided in the Appendix (Section 5.6). For further details, please refer to [305, 306].

5.3.3 Adaptive Hill-Climbing-Based Patrol Scheduling (AHBPS)

Table 5.2: Notations used in AHBPS

Notation	Description
$G(N, E)$	Graph with nodes N and edges E
C	Set of vehicles
D	Set of depots
S	Set of shifts
T^{shift}	Duration of a shift
T^{patrol}	Duration of each patrol
$W_{l,l'}$	Travel time between locations l and l'
$Visited_{locations}$	Locations that have been visited
$routes$	Collection of routes for each vehicle and shift
$L_{l,s}$	Last visit time of location l in shift s
LL	List of locations to be revisited
SST	Start time of a shift
SET	End time of a shift
CT	Current time during the patrol
CS	Current solution in the hill-climbing optimization process
CSO	Objective value of the current solution
CSO_{better}	Improved objective value of a neighboring solution
C_{route}	Currently selected route for a patrol
$fluctuation$	Random fluctuation applied to travel time

In this section, we introduce a heuristic algorithm designed to address both small and large instances of the NP-hard VPS problem efficiently. The proposed AHBPS method,

Algorithm 31 The Heuristic Method: AHBPS

Require: $G(N, E)$, C, D, S, T^{shift}, T^{patrol}, $W_{l,l'}$
Ensure: $Visited_{locations}$, $routes[s][c]$

1: Initialize graph G, vehicles C, depots D, shifts S, shift time T^{shift}, patrol time T^{patrol}, and travel times $W_{l,l'}$
2: $Visited_{locations} \leftarrow$ RouteFinding$(G, C, D, S, T^{shift}, T^{patrol}, W_{l,l'})$
3: $L_{l,s}, routes \leftarrow$ ShiftManagement$(G, C, D, S, T^{shift}, T^{patrol}, W_{l,l'})$
4: **return** $Visited_{locations}$, $routes[s][c]$

presented in Alg.31, comprises two primary components: the *Routing Process* and the *Patrol Scheduling Process*. The *Routing Process*, detailed in Alg.32, identifies the most efficient patrol routes and vehicle shifts, while the *Patrol Scheduling Process*, outlined in Alg. 33, ensures that vehicles adhere to an optimized schedule to maximize coverage efficiency.

The AHBPS method (Alg. 31) takes as inputs the graph $G(N, E)$, the vehicle fleet C, depot location D, number of shifts S, maximum operation time per shift T^{shift}, minimum patrol time T^{patrol}, and travel times $W_{l,l'}$. The algorithm outputs the visited locations and routes (paths) for vehicles.

To determine patrol routes, AHBPS invokes the `RouteFinding` function in line 3, which is detailed in Alg. 32. This function outputs the set of patrolled locations, $Visited_{locations}$, based on the optimal solution. The process begins by simulating travel time fluctuations to reflect real-world conditions, introducing random variations in travel times $W_{l,l'}$ between locations l and l' (lines 2-5). The primary optimization method employs Hill Climbing (lines 6-10), where neighboring solutions are generated from the current solution (line 8). If a neighboring solution improves the objective, it is selected as the new current solution (lines 9-10). This iterative process continues until no further improvement is possible. Finally, for each location in the current solution (CS), if it meets the optimal criteria, it is added to the $Visited_{locations}$ set. Alg. 32 balances shift constraints, travel times, and real-time fluctuations, ensuring efficient and dynamic patrol routes to enhance overall patrol coverage.

AHBPS then invokes the `ShiftManagement` function (line 4) to ensure all locations are visited according to the planned vehicle routes. The patrol scheduling process is detailed in Alg. 33, which aims to enable vehicles to follow an optimal schedule across multiple shifts. The process begins by initializing key parameters, such as the current time (CT), the last visit time for each location ($L_{l,s}$), and the patrol routes. The algorithm ensures that no location remains unpatrolled within a given interval by using the `Revisit` function (lines

Algorithm 32 Routing Process

```
function RouteFinding(G, C, D, S, T^shift, T^patrol, W_{l,l'})
    if a random value < 0.25 then
        for all (l, l') ∈ G do
            fluctuation ← random value of ±2
            W_{l,l'} ← W_{l,l'} + fluctuation
        end for
    end if
    while Improvement found do
        neighbors ← Generate neighboring solutions from CS
        CSO_better ← Evaluate and select the best neighbor using updated W_{l,l'}
        if CSO_better > CSO then
            CS ← Update Current Solution
            CSO ← CSO_better
        end if
    end while
    for all location ∈ CS do
        if location is visited in the best solution then
            Visited_locations ← Visited_locations ∪ {location}
        end if
    end forreturn Visited_locations
end function
```

2-6).

The `ShiftManagement` function (lines 7-18) handles vehicle route selection during each shift. For every vehicle in each shift, if a location requires revisiting, the algorithm selects the next optimal location for the vehicle based on adjacency and travel time. This ensures that all locations are patrolled within the allocated shift time T^{shift}. Routes are dynamically updated with each visit (line 11). If a revisit is required, parameters such as travel time $travel_{time}$, current time CT, and the routes for each shift and vehicle ($routes[s][c]$) are updated accordingly. The last visit time $L_{l,s}$ for each location is also recorded (line 16). Alg. 33 ensures that each location is patrolled within the required time frame, minimizing gaps between visits and optimizing vehicle routes for efficient coverage during patrol shifts. The final output of AHBPS (line 5 in Alg. 31) includes the set of visited locations and the routes taken by the vehicles in each shift, providing a comprehensive patrol plan.

Theorem 2 demonstrates the time complexity of AHBPS, analyzing each function to validate its computational efficiency.

Algorithm 33 Patrol Scheduling Process

Require: $CT, L_{l,s}, LL, SST, SET, routes, CS, CSO$

```
function Revisit(CT, L_{l,s}, LL)
    if (CT − L_{l,s}) ≤ 30 AND (!LL OR LL ≤ CT) then return True
    elsereturn False
    end if
end function
function ShiftManagement(S, C, L_{l,s}, LL, SST, SET, routes, CS, CSO, W_{l,l'})
    for all s ∈ S do
        for all c ∈ C do
            while CT + T^{patrol} ≤ SET do
                c_{route} ← Select the next best location
                if Revisit(CT, L_{l,s}, LL) then
                    travel_{time} ← W_{l,l'}[current_{location}, c_{route}]
                    CT ← CT + travel_{time} + T^{patrol}
                    routes[s][c] ← Update the route with the visited location
                    L_{l,s} ← Update Last Visit Time for the location
                end if
            end while
        end for
    end for
    routes[C] ← D return L_{l,s}, routes[s][c]
end function
```

Theorem 2. *The time complexity of the AHBPS algorithm is* $O(|S| \cdot |C| \cdot |N| \cdot I)$

Proof. To analyze the time complexity of each algorithm, we break down the operations within each function and examine their individual complexities.

The `AHBPS` function serves as the main function, responsible for initializing parameters and invoking two key functions: `RouteFinding` and `ShiftManagement`. Since the time complexities of `RouteFinding` and `ShiftManagement` are the dominant contributors to the overall runtime of `AHBPS`, its total complexity can be expressed as $O(RouteFinding) + O(ShiftManagement)$

The `RouteFinding` function can be broken down as follows: The initial random value check is a constant-time operation, resulting in a complexity of $O(1)$. If the random value check is satisfied, a loop runs over all edges $(l, l') \in G$, adjusting the weight $W_{l,l'}$ with a

constant-time operation. Letting $|E|$ represent the number of edges in G, this loop has a time complexity of $O(|E|)$.

The main iterative process seeks improved solutions by generating neighboring solutions and evaluating them. Assuming this loop runs N times, where N represents the number of locations, the time complexity depends on the effort required to evaluate neighbors and select the best solution. Let I denote the complexity of evaluating neighbors, which corresponds to the number of iterations. Consequently, this component's complexity is $O(N \cdot I)$.

Finally, the algorithm iterates over each location in the current solution CS, which may have up to $|N|$ locations, where $|N|$ is the number of nodes in G. Thus, this step has a complexity of $O(|N|)$. Combining all components, we obtain the overall time complexity for `RouteFinding`:

$$\text{Time Complexity of RouteFinding} = O(|E| + N \cdot I + |N|)$$

The `ShiftManagement` function consists of nested loops and calls the `Revisit` function. Its time complexity is derived as: The first two loops iterate over the sets of shifts S and vehicles C, giving a complexity of $O(|S| \cdot |C|)$. The inner `While` loop executes as long as the condition $\text{CT} + T^{patrol} \leq \text{SET}$ holds. Inside the loop, it selects the next best location involving a selection or optimization process, which has a complexity of $O(I)$, where I is the number of Iterations. The `Revisit` function operates in constant time, with a complexity of $O(1)$. Assuming the `While` loop runs N number of times for each shift-vehicle pair, the inner complexity is $O(N \cdot I)$. Combining all parts, we obtain the time complexity of `ShiftManagement` as $O(|S| \cdot |C| \cdot |N| \cdot I)$ Thus, AHBPS is dominated by `ShiftManagement`, and the time complexity of AHBPS is $O(|S| \cdot |C| \cdot |N| \cdot I)$ □

In the next section, we introduce an alternative heuristic approach based on a Genetic Algorithm, referred to as GDVPS.

5.3.4 Genetic-Based Dynamic Vehicle Patrol Scheduling (GDVPS)

To build upon the results achieved by the AHBPS algorithm, we propose a robust meta-heuristic approach based on Genetic Algorithms (GAs), referred to as GDVPS. Fig. 5.2 provides a comprehensive overview of the proposed GDVPS framework in this study. The GDVPS algorithm enhances the initial patrol schedules generated by AHBPS. Leveraging an evolutionary process, it refines the patrol routes to achieve an optimal or near-optimal schedule for the vehicles.

Genetic Algorithms (GAs) are powerful optimization techniques inspired by the principles of natural selection and genetics. These methods address complex optimization problems by evolving a population of potential solutions over successive generations. Through the processes of selection, crossover, and mutation, GAs iteratively improve solutions, ultimately aiming to identify the most effective outcome [307, 308].

GDVPS begins by generating an initial population of patrol schedules. A portion of this population is seeded with the outputs from AHBPS to provide a strong starting point, while the remaining schedules are generated randomly to promote diversity and ensure the exploration of a wide range of routing possibilities. Each schedule in the population represents a complete set of routes and timings for all vehicles across their assigned shifts.

Figure 5.2: GDVPS Flowchart

The fitness of each schedule is evaluated based on its effectiveness in maximizing the

number of unique locations visited. This evaluation ensures that the algorithm aligns with the primary objective of covering as many locations as possible. Subsequently, the top-performing schedules are selected and combined to generate new schedules. This process, known as crossover, allows the algorithm to merge and refine the most effective components of different schedules, progressively enhancing the patrol routes.

To further diversify the solutions and prevent convergence to a suboptimal result, the algorithm introduces small random modifications (mutations) to the schedules. These mutations may involve altering the order in which locations are visited by each vehicle or making similar adjustments, fostering exploration and innovation in the solution space.

GDVPS iteratively applies selection, crossover, and mutation over multiple generations, continuing until either a predefined number of generations is reached or no further improvements are observed (stagnation). The final output is an optimized patrolling schedule that maximizes the number of unique locations visited while adhering to all constraints. To maintain solution quality, GDVPS incorporates elitism by retaining the top 5% of the population across generations. These elite individuals are preserved and combined with new offspring to form the next generation. Fitness progress is monitored throughout, with mutation rates dynamically adjusted based on the observed improvements, ensuring a balance between exploration and exploitation.

What sets GDVPS apart from other heuristic methods, such as Ant Colony Optimization (ACO), is its tailored mutation and crossover strategies. These genetic operators are specifically designed to address the unique constraints of the Vehicle Patrol Scheduling (VPS) problem. This customization allows GDVPS to generate feasible solutions that satisfy all problem requirements, a task that general heuristic methods might struggle to achieve.

5.3.5 Performance Evaluation

In this section, we evaluate the proposed methods—OP, AHBPS, and GDVPS—on both small and large complex networks. The primary performance metric used is the number of unique locations visited (total visits).

The graphs used for testing were generated using real-world maps extracted from Open Source Maps (OSM). By assigning varying numbers of nodes to represent target locations (e.g., banks) on the same map size, we created diverse graph instances with different configurations. This allowed us to simulate a variety of real-world scenarios to test the three approaches comprehensively. Using the NetworkX and OSMNX libraries in Python [309, 310], we processed the extracted maps and converted them into graph structures to

serve as inputs for the algorithms. An example of one such generated graph is shown in Fig. 5.4, which corresponds to the map displayed in Fig. 5.3.

Travel times between nodes were calculated based on the distances in the generated maps. To simulate real-world traffic conditions, weights were assigned to the links using a normal distribution with a mean travel time of 15 minutes and a standard deviation of 5 minutes. Additionally, real-time dynamics, such as varying traffic density, were incorporated by adjusting travel times by ± 2 minutes in 25% of the simulation runs. These adjustments account for scenarios like reduced travel time during low traffic or increased travel time due to incidents, providing a realistic evaluation of the algorithms' performance under dynamic conditions.

To ensure reliable results, the simulations were run with initial base parameter values, and results were analyzed across multiple instances. Subsequently, parameters were varied to observe their influence on performance. Unless otherwise stated, the default values were as follows: rest time was set to 10 minutes, patrol time per location was 5 minutes, and the mean travel time was 15 minutes. The results were averaged over 100 simulation runs to ensure accuracy and mitigate biases that might arise from individual runs. A summary of the simulation parameters and their default values is provided in Tab. 5.3.

Figure 5.3: An example of a map of Waterloo, Ontario, Canada used in our simulations – Extracted with OSMNX Python

Figure 5.4: Extracted graph from Fig. 5.3; Generated by NetworkX and OMSNX Python – Weights are representing travel time between each pair of nodes

Table 5.3: Parameters and their Values used in our simulations

AHBPS Variables	
Rest Time in the Depot	10 minutes
Patrol Time for locations in each visit	5 minutes
Revisit Gap for locations	30 minutes
Travel Time between each node (if exist)	An adaptive random number chosen from a normal distribution with mean 15 and standard deviation of 5

Evaluation Over Small Networks

In this subsection, we evaluate the performance of our proposed methods, OP, AHBPS, and GDVPS, on small networks. We compared the optimal solution (OP) against our proposed heuristics across various scenarios, with the number of vehicles ranging from `1-4` and the number of locations set to `5, 10, 15`. The evaluation was conducted under different conditions, as illustrated in Fig. 5.5. In the figures, "V" represents the number of vehicles, and "L" represents the number of locations.

Figure 5.5: Performance comparison (total visits) of OP, AHBPS, and GDVPS methods under different scenarios

Fig. 5.5 presents a comparative analysis of the three methods, OP, AHBPS, and GDVPS, evaluating their effectiveness in terms of the number of locations visited across various scenarios with different numbers of vehicles and locations. The OP method consistently achieves the highest number of visited locations across all scenarios, as expected. Designed to provide the most optimal outcome for each scenario, OP outperforms both AHBPS and GDVPS in every case. For instance, in the scenario with 2 vehicles and 15 locations (2V, 15L), OP successfully visited and re-visited all locations, whereas AHBPS achieved a total of 14 visits (including revisits), and GDVPS achieved 16 visits. While AHBPS and GDVPS are heuristic methods and do not guarantee optimality, their performance demonstrates reasonable proximity to the optimal solution. The OP method, on the other hand, excels by thoroughly exploring all possible configurations, pruning suboptimal regions, and ensuring that the best solution is identified. This guarantees the highest level of efficiency and effectiveness for the given scenarios.

Fig. 5.6 depicts the execution times for the proposed methods. The OP approach exhibits significantly longer execution times due to its exhaustive search, with execution time increasing exponentially as the network size grows. For instance, in a small network with 15 locations and a fleet of 4 vehicles, OP required over 4000 seconds to execute. In contrast, AHBPS maintains a consistent execution time of approximately 0.01 seconds across all instances, showcasing its efficiency. GDVPS, while slightly more time-consuming,

Figure 5.6: Execution time comparison between OP, AHBPS and GDVPS

averages around 0.3 seconds, demonstrating a reasonable trade-off between performance and computation time. The OP approach becomes impractical for instances involving more than 15 locations due to excessive computation times, highlighting the challenges posed by the NP-hard nature of the problem. These results indicate that, while OP provides the most theoretically optimal solutions, it is infeasible for larger problem sizes. GDVPS emerges as a more scalable and robust alternative, particularly in complex scenarios, offering a balance between execution time and solution quality. Although AHBPS remains competitive in simpler cases, its effectiveness diminishes as problem complexity increases.

Fig. 5.5 and Fig. 5.6 provide several key insights. First, OP consistently outperforms the heuristic approaches in terms of the total visits metric. However, the use of OP is often impractical due to its high computational demands, particularly as the problem size grows. Second, GDVPS demonstrates superior scalability and robustness compared to AHBPS, especially in complex scenarios with an increasing number of vehicles and locations. Third, OP becomes virtually unusable for instances involving more than 15 locations, as its execution time grows prohibitively large. Finally, while AHBPS remains competitive in simpler scenarios, its performance significantly deteriorates compared to GDVPS and OP as the problem complexity increases, highlighting its limitations in handling larger and more intricate networks.

Evaluation Over Large Networks

In this subsection, we evaluate the performance of our proposed heuristic methods, AHBPS and GDVPS, on large networks by varying multiple parameters. Specifically, we assess different scenarios by altering the number of locations and the size of the patrol vehicle fleet. The evaluations were conducted with fleets comprising 10, 12, 15, and 20 vehicles, while the number of nodes varied between 500, 750, and 1000.

Figure 5.7: Total visits comparison of AHBPS and GDVPS methods under different scenarios

Fig. 5.7 illustrates the comparison between the proposed heuristics in terms of total visits.

The results clearly show that GDVPS consistently outperforms AHBPS in nearly all instances, particularly as the number of locations increases. This highlights GDVPS's superior scalability with increasing problem complexity. While both methods exhibit improvements in visiting more locations as the problem size grows, the performance gap widens significantly in favor of GDVPS. For example, in a scenario with 20 vehicles and 1000 locations, GDVPS successfully covered approximately 80% of the locations, whereas AHBPS managed to cover only about 45%. The superior performance of GDVPS can be attributed to the genetic algorithm's ability to efficiently explore and exploit the search space through mechanisms like crossover and mutation. These techniques enable GDVPS to discover optimal or near-optimal solutions more effectively, making it better suited for

Figure 5.8: Execution time comparison between AHBPS and GDVPS

handling complex and large-scale problems. Despite this, AHBPS remains a competitive heuristic approach, particularly in simpler scenarios. Its relatively strong performance is due to the Hill-Climbing strategy it employs, which helps it achieve reasonable solution quality even with its simpler optimization framework.

Figure 5.9: Total traveled distance (in kilometers) comparison between AHBPS and GDVPS

The execution times for different scenarios are shown in Fig. 5.8. As expected, AHBPS is faster at obtaining suboptimal solutions compared to GDVPS, owing to the factors discussed earlier in this chapter. GDVPS consistently takes longer to run than AHBPS. The execution time for AHBPS remains nearly constant across all scenarios, demonstrating its robustness against changes in problem size or complexity. This characteristic makes AHBPS particularly appealing when rapid implementation is a priority. On the other hand, GDVPS, while slower, exhibits a linear growth in execution time as problem size increases. For example, GDVPS requires approximately 120 seconds for instances involving 1000 locations. Despite the longer runtime, GDVPS delivers significantly better coverage, as reflected in its ability to visit more locations. The linear growth trend, as shown in purple, underscores its scalability and effectiveness for larger problem instances.

While AHBPS excels in speed, its simplicity limits its ability to thoroughly explore the solution space, making it less effective than GDVPS at optimizing total visits. However, in scenarios where sudden changes necessitate a rapid generation of new schedules, AHBPS proves ideal, producing results in under a second. It is crucial to weigh the trade-off between speed and solution quality, as prioritizing speed alone can compromise accuracy and coverage.

Fig. 5.9 illustrates the total distance traveled (in kilometers) across various scenarios. As depicted, an increase in the number of vehicles or locations leads to a correspond-

ing rise in traveled distance. This can be attributed to two primary reasons: first, with more vehicles in the simulations, there are additional routes being traversed; second, an increase in available locations provides vehicles with more destinations to visit, thereby resulting in greater distances traveled. The trends for each algorithm across the scenarios are highlighted using different colors. For example, in the scenario with 15 vehicles and 1000 locations, the total traveled distance for GDVPS was approximately 850 kilometers over the entire operation time of 12 hours. In contrast, the total distance for AHBPS was around 625 kilometers. This difference is explained by GDVPS's ability to cover more locations, which naturally requires vehicles to travel greater distances to reach and service those locations.

Parameters Engineering

As we have seen through all the simulations and different tested scenarios, we can conclude that the proposed GDVPS algorithm is highly effective in both small and large networks, thanks to its robust implementation. Additionally, it demonstrates considerable efficiency in real-time scenarios, as its execution time remains relatively fast.

In this subsection, we analyze the effects of varying different parameters on the total visits using GDVPS. The evaluation focuses on large networks, specifically the scenario with 10 vehicles and 1000 locations. To better understand the impact of these parameter variations, the results are normalized to a scale of 0–100, providing clearer insights. It is important to note that a value of 100 does not indicate perfect total coverage; rather, it represents the best-achieved result among the cases being compared.

Figure 5.10: Total visits comparison between the proposed methods with different patrol time

Figure 5.11: Total visits comparison between the proposed methods with different rest period

We begin by analyzing the effects of varying patrol durations for each location. Three values were considered: the initial value of 5 minutes, along with 10 and 15 minutes. As

expected, increasing the patrol time for each location results in reduced overall coverage. This reduction occurs because vehicles are required to spend more time, 10 or 15 minutes at each location, which decreases the available time to visit additional locations over the operational period. Fig. 5.10 provides a comprehensive visualization of the impact of varying patrol durations on total coverage.

Fig. 5.11 illustrates the effects of varying the rest period at the depot for vehicles. The results reveal a minor improvement in coverage when the rest time is reduced from 10 minutes to 5 minutes, with an increase of approximately 4 units for GDVPS and around 5 units for AHBPS. As expected, a rest time of 15 minutes resulted in lower performance compared to shorter rest periods, as vehicles and crews spend more time on breaks, reducing the time available to cover additional locations. However, it is important to note that shorter rest periods could potentially impact psychological and physical well-being, factors that were not evaluated in this study.

As previously mentioned, the travel time between nodes is calculated using real-world maps. The observed travel times follow a normal distribution with a mean of 15 and a standard deviation of 5. In this section, we examine the effects of changing the mean travel time to 10 and 20. When the mean travel time is reduced to 10, we anticipate increased coverage, while a mean of 20 is expected to result in decreased coverage due to longer travel times between nodes. As shown in Fig. 5.12, this trend aligns with expectations, as shorter travel times allow vehicles to cover more locations. In conclusion, shorter travel times contribute to increased coverage, confirming our predictions.

Lastly, we evaluated the effects of varying operation times on total visits. Specifically, we compared two different values (10 and 8 hours) against the base value of 12 hours. As expected, a 12-hour operation time results in better coverage due to the fleet having more time to secure locations. Fig. 5.13 illustrates these changes, showing a similar trend for both methods. In conclusion, GDVPS demonstrates high effectiveness in both small and large networks, offering near-optimal performance across all tested scenarios. Additionally, its moderate execution time makes it efficient for real-time applications.

5.4 Reinforcement Learning & Heuristic Approach for VPS with Emergency Cases

In this section, we address the same VPS problem with an update. The update is that there are emergency cases that vehicles must address those with the upmost priority. We start by introducing the problem formulation in the next subsection.

Figure 5.12: Performance comparison (total visits) between the proposed methods with different travel time distributions (standard deviation is fixed to 5)

5.4.1 System Model and Problem Definition

In this section, we explain the system model used in this study, detailing its structure and main components. The proposed model is designed to handle VPS in dynamic environments by integrating constraints such as operational time limits, required rest periods, location revisit intervals, and emergency response requirements. Table 5.4 gives a complete overview of the notations and definitions used in our study. The system represents the patrol area as a graph, where nodes correspond to locations requiring patrols, and edges denote the travel paths between them. Vehicles are assigned to patrol routes while optimizing coverage efficiency, ensuring timely responses to emergencies, and maintaining operational feasibility under predefined constraints. To achieve this, our model incorporates real-time adaptability, allowing schedules to be dynamically updated in response to changing conditions, such as emergency demands or traffic variations. This structure enables scalable and efficient patrol planning, ensuring both broad coverage and rapid incident response in complex urban environments.

As illustrated in Fig. 5.14, we conceptualize the operational environment for a fleet of patrol vehicles as a weighted directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ represents the set of nodes, which includes a depot $l \in \mathcal{N} = 0$, and a set of locations $l = \{1, 2, ..., |\mathcal{N}|\}$ that must be patrolled regularly. Each location has a minimum patrolling time $\mathcal{T}^{patrol}$ and a maximum allowable gap between visits $\mathcal{T}^{gap}$. The set $\mathcal{E}$ signifies the links representing

Figure 5.13: Total visits comparison between the proposed methods with different operation time

possible routes between these nodes, with weights corresponding to travel time between locations, denoted as $\mathcal{W}_{l,l'}$, which are measured based on current traffic conditions. These weights are dynamically updated to reflect the real-world nature of the problem. During the operational period, vehicles in the fleet ($c \in \mathcal{C}$) are systematically assigned routes within the graph. Each route begins at the depot, visits designated locations, and returns to the depot upon completing its shift. The complexity of this routing lies in the possibility of revisiting certain locations multiple times, as the vehicle may have sufficient time during its shift to conduct additional patrols, ensuring continuous safety and optimal coverage. Each vehicle must adhere to a mandatory rest time $\mathcal{T}^{rest}$ after a maximum operation time of $\mathcal{T}^{shift}$. This structure ensures comprehensive coverage of all areas of interest during the operational time.

To manage the entire patrol system, we have established the Central Fleet Control Unit (CFCU) at the depot. The CFCU oversees all vehicles, updates their schedules in real time, and dispatches them to handle emergencies as they arise. By analyzing live data such as vehicle locations, travel times, and emergency requests, the CFCU ensures that the patrol system can adapt to the ever-changing urban landscape while providing effective coverage and response times.

Our goal is to enhance the total number of locations visited during the workday to improve coverage. To accomplish this, we make sure that no more than one vehicle is assigned to the same location at the same time and that each location is revisited within

Table 5.4: Notation Definitions

Notation	**Definition**
$\mathcal{G} = (\mathcal{N}, \mathcal{E})$	Graph representing the patrol area, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges (routes between locations).
$\mathcal{N}$	Set of nodes, including the depot and patrol locations.
$l \in \mathcal{N}$	A location (node) that must be patrolled, where $l = 0$ represents the depot.
$\mathcal{E}$	Set of edges representing possible travel routes between locations.
$\mathcal{W}_{l,l'}$	Weight of edge between locations l and l', representing travel time between them based on real-world traffic conditions.
$\mathcal{C}$	Set of vehicles assigned to patrol the area.
$c \in \mathcal{C}$	A specific vehicle in the fleet.
$\mathcal{T}^{patrol}$	Minimum patrolling time required at each location.
$\mathcal{T}^{gap}$	Maximum allowable time gap between consecutive visits to a location.
$\mathcal{T}^{shift}$	Maximum operation time for each vehicle before returning to the depot.
$\mathcal{T}^{rest}$	Mandatory rest period for vehicles after completing a shift.
CFCU (Agent)	Central Fleet Control Unit responsible for dispatching and route optimization.
$\mathcal{P}^{emergency}$	Likelihood of an emergency occurring at a specific location, requiring immediate response.

a designated timeframe to ensure ongoing safety. There is a likelihood of emergency situations (e.g., robbery) occurring at specific locations during shifts. In such cases, the nearest vehicle must be dispatched to the emergency site, necessitating a complete reorganization of that vehicle's schedule and the entire fleet.

As illustrated in Fig. 5.14, consider a scenario in which Vehicle B is currently patrolling Location 3, while Vehicle A is en route to patrol Locations 1 or 2. Location 4, which was previously patrolled by Vehicle B, experiences an emergency situation requiring immediate attention. In this case, the CFCU must dynamically reassign a vehicle to address the emergency at Location 4. Based on real-time data, such as the current positions of the vehicles and their travel times to Location 4, the CFCU evaluates the optimal assignment.

Figure 5.14: Illustration of the system model: The operational environment for the fleet of patrol vehicles is represented as a graph. Locations are numbered and differentiated by distinct colors, with nodes representing patrol locations and the depot serving as the starting and ending points for vehicles. Edges denote travel paths between nodes, with weights corresponding to travel times. The CFCU is responsible for monitoring vehicle routes, dispatching vehicles, and dynamically updating schedules in response to real-time changes.

It may direct Vehicle B to interrupt its ongoing patrol at Location 3 and proceed to Location 4, or alternatively, it could dispatch Vehicle A to Location 4. To ensure efficient emergency handling, the CFCU simultaneously reorganizes the routes and schedules of the entire fleet, balancing the need to respond promptly to the incident with maintaining overall patrol coverage.

To illustrate the computational complexity of the VPS problem, we can compare it to the well-known Traveling Salesman Problem (TSP) [304]. The TSP requires finding a Hamiltonian cycle that minimizes the total travel distance while visiting each city exactly once and returning to the starting point. Although the VPS problem has similarities with TSP in terms of covering locations, it introduces additional complexities, such as flexibility in the number of locations visited, which is influenced by operational constraints. Despite these differences, the structural similarity between determining an optimal patrol route in VPS and constructing a Hamiltonian cycle in TSP indicates that the VPS problem is computationally demanding. By considering the possibility of reducing an instance of TSP to an instance of VPS in polynomial time, we emphasize the inherent complexity of the VPS problem. This highlights the need for advanced algorithmic solutions to effectively address the issue, particularly in the realm of dynamic patrol scheduling and route optimization.

In the following sections, we will concentrate on developing robust strategies to tackle these challenges and create near-optimal patrol plans for complex scenarios.

5.4.2 The Heuristic Method: AHBPS2

In this section, we present the Adaptive Hill-Climbing-Based Patrol Scheduling (AHBPS2) algorithm (see Alg. 34), which aims to effectively manage vehicle routes while ensuring a quick response to emergencies. The algorithm works by iterating through scheduled shifts and utilizing a hill-climbing approach to optimize patrol routes, thus guaranteeing thorough area coverage and swift emergency response in dynamic settings. The inputs for AHBPS2 consist of the graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$, the vehicle fleet $\mathcal{C}$, the depot location $\mathcal{D}$, the set of shifts $\mathcal{S}$, the maximum operation time per shift $\mathcal{T}^{\text{shift}}$, the minimum patrol time $\mathcal{T}^{\text{patrol}}$, the travel times $\mathcal{W}_{l,l'}$, and the probability of an emergency $\mathcal{P}^{\text{emergency}}$. The algorithm outputs the set of visited locations, the routes for each vehicle during each shift, and a record of handled emergencies.

The algorithm starts by setting key parameters, including the shift start time ($\mathcal{SST}$), shift end time ($\mathcal{SET}$), current time ($\mathcal{CT}$), a list of visited locations ($Visited_{locations}$), and a data structure to keep track of the route of each vehicle ($routes$) in Line 1. For each shift $s \in \mathcal{S}$, the start and end times are defined (Line 3). Then, a while-loop (Line 4) runs, continuing until the current time $\mathcal{CT}$ hits the shift end time $\mathcal{SET}$.

During each iteration of the while-loop, the algorithm examines every vehicle $c \in \mathcal{C}$ (Line 5). For each vehicle, it first checks for emergencies by calling the utility function `CheckForEmergencies` with inputs $\mathcal{C}$ and $\mathcal{P}^{\text{emergency}}$ (Line 6). if emergencies are detected (Line 7), the algorithm calls the `HandleEmergencies` function to update vehicle states and routes accordingly (Line 8). After that, if the current vehicle is available and not involved in an active emergency (Line 9), the next patrol location is identified by using the utility function `PlanNextLocation` with the vehicle c and the current time $\mathcal{CT}$ (Line 10). The vehicle's location is then updated to this new spot (Line 11), the location is noted in the list of visited locations (Line 12), and it is added to the vehicle's route for the current shift (Line 13).

After processing all vehicles in the inner loop, the algorithm implements hill-climbing optimization on the current set of routes for the shift (Line 14). The current time $\mathcal{CT}$ is then updated to move on to the next iteration (Line 15). Once the while-loop has been completed for all shifts, the algorithm outputs the full set of visited locations, the routes for each vehicle per shift, and the emergency log (Line 14).

The utility functions outlined in Alg. 35 are essential for the functioning of AHBPS2. In

particular, the `CheckForEmergencies` function (called at Line 6 in Alg. 34) simulates emergencies based on the given probability; the `HandleEmergencies` function (called at Line 8 in Alg. 34) modifies vehicle states and routes to ensure effective emergency response; and the `PlanNextLocation` function (called at Line 10 in Alg. 34) determines the next patrol location based on the current patrol strategy and time constraints. The `PlanNextLocation` function works by first using the current simulation time ($\mathcal{CT}$) along with the vehicle's current state to calculate a destination that aligns with the overall patrol strategy. In practice, this involves evaluating factors like the order of locations to be covered and ensuring that the chosen location can be reached within the remaining time available in the vehicle's shift. If the vehicle is operating under normal conditions (i.e., no emergency is being handled), the function selects the next point in the pre-determined patrol sequence. However, if the vehicle is flagged as "responding to an emergency," the function immediately updates its destination to the emergency location, overriding the routine patrol plan. This dual approach ensures that the system remains efficient in routine coverage and adaptive in crisis situations. Together, these elements ensure that AHBPS2 is both adaptive and efficient, balancing routine patrol scheduling with the necessary flexibility for real-time emergency responses.

Algorithm 34 Adaptive Hill-Climbing-Based Patrol Scheduling (AHBPS2)

Require: $\mathcal{G}(\mathcal{N}, \mathcal{E})$, $\mathcal{C}$, $\mathcal{D}$, $\mathcal{S}$, $\mathcal{T}^{shift}$, $\mathcal{T}^{patrol}$, $\mathcal{W}_{l,l'}$, $\mathcal{P}^{\text{emergency}}$
Ensure: $Visited_{locations}$, $routes$, $emergencies$

```
Initialize SET, SST, CT, Visited_locations, routes ← ∅
for all s ∈ S do
    Set start time SST and end time SET for s
    while CT < SET do
        for all c ∈ C do
            emergencies ← CheckForEmergencies(C, P^emergency)
            if emergencies are not empty then
                Call HandleEmergencies(emergencies, C) to update vehicle states
            end if
            if c is available and no active emergency then
                next_location ← PlanNextLocation(c, CT)
                Update c's location to next_location
                Add next_location to Visited_locations
                Append next_location to routes[s][c]
            end if
        end for
        Apply hill-climbing optimization
        Update CT for the next iteration
    end while
end for
    return (Visited_locations, routes, emergencies)
```

Algorithm 35 Utility Functions for AHBPS2

```
function CheckForEmergencies(C, P^emergency)
    emergencies ← ∅
    for all c ∈ C do
        if random event triggers with probability P^emergency then
            Record emergency details for vehicle
            Add emergency details to emergencies
        end if
    end for
    return emergencies
end function
function HandleEmergencies(emergencies, C)
    for all emergency ∈ emergencies do
        Assign the nearest available vehicle to handle the emergency
        Update vehicle's status to 'handling emergency'
        Update vehicle's route to emergency location
    end for
    return C
end function
function PlanNextLocation(c, CT)
    Calculate the next location based on patrol strategy and CT
    if vehicle c is responding to an emergency then
        Update its location to the designated emergency destination
    end if
    return next location
end function
```

5.4.3 The Reinforcement Learning Approach: $\mathcal{P}^4\mathcal{O}$

To enhance the performance of the AHBPS2 algorithm and to adapt to the dynamic nature of the environment, we propose an RL approach called Patrol Planning with Proximal Policy Optimization algorithm ($\mathcal{P}^4\mathcal{O}$). We frame the VPS problem as an MDP, which enables us to clearly define the states, actions, transitions, rewards, and objectives within the patrol scheduling framework [311]. The MDP is structured as a tuple $(\mathcal{S}, \mathcal{A}, \gamma, \mathcal{P}, \mathcal{R})$ [3]. The functions for state, action, and reward within the MDP components of our problem are outlined as follows:

Agent: The agent is CFCU, situated at the depot. Its main responsibility is to deter-

mine the best routes for vehicles and the locations each vehicle needs to visit. Additionally, it must reorganize the fleet in emergencies and dispatch the nearest vehicle(s) to secure high-risk locations, regardless of the circumstances.

State: The state space encompasses the various situations or conditions the agent must evaluate for each vehicle. A state is characterized by five key attributes: (1) the set of locations each vehicle has visited up to timestep t, (2) the current shift number, which indicates how many shifts each vehicle has completed, (3) the emergency status, highlighting location(s) that need immediate attention, (4) the last visit time for each location, and (5) the remaining shift time for each vehicle, which can vary since vehicles may be dispatched at different times by the agent. This state space combines both discrete and continuous elements. The discrete components include the set of visited locations, the shift number, and the emergency status, while the continuous components consist of the remaining shift time and the last visit time for locations. This mixed representation effectively captures the dynamic and varied nature of the patrol scheduling problem, allowing the agent to make well-informed decisions based on both categorical and numerical data.

Actions: The actions available to the agent involve selecting the next location for a vehicle to visit based on its current state. The options are restricted to locations that are accessible and not blocked. This action space is discrete as it requires choosing a location from a defined set of available options.

Reward: The reward function is designed to prioritize patrolling a greater number of locations. Visiting a location yields a small positive reward (+0.5) to encourage increased coverage. Conversely, if an emergency occurs and the agent fails to dispatch a vehicle within the allowable response time, which is determined by the travel time from the nearest vehicle's position to the emergency location, a substantial penalty (-50) is imposed.

By framing the problem within the MDP framework, the agent learns how to effectively assign vehicles to various locations in the environment to maximize total visits. This is achieved by taking actions that lead to more advantageous states, thereby accumulating higher rewards. The resulting policy will outline optimal routes for each vehicle, ensuring they can respond to emergencies efficiently.

The details of the $\mathcal{P}^4\mathcal{O}$ algorithm are outlined in Alg. 36. The process starts with the initialization phase, where the environment is set up with the network graph $\mathcal{G}$, the vehicle set $\mathcal{V}$, and temporal constraints such as the maximum shift duration $\mathcal{T}^{shift}$, the required rest period $\mathcal{T}^{rest}$, and the minimum patrol time $\mathcal{T}^{patrol}$. Additionally, the travel time matrix $\mathcal{W}_{l,l'}$ and the emergency probability $\mathcal{P}^{emergency}$ are established (Line 1). Next, in Line 2, the $\mathcal{P}^4\mathcal{O}$ agent is initialized using an actor-critic network.

Training occurs over $\mathcal{K}$ episodes, which are chosen based on empirical methods to ensure

Algorithm 36 Patrol Planning with Proximal Policy Optimization ($\mathcal{P}^4\mathcal{O}$)

Require: $\mathcal{G} = (\mathcal{N}, \mathcal{E})$: Network graph with nodes $\mathcal{N}$ (locations) and edges $\mathcal{E}$; $\mathcal{V}$: Set of vehicles; $\mathcal{T}^{shift}$: Maximum shift duration; $\mathcal{T}^{rest}$: Required rest period between shifts; $\mathcal{T}^{patrol}$: Minimum patrol time per location; $\mathcal{P}^{emergency}$: Probability of emergency; $\mathcal{W}_{l,l'}$: Travel time matrix between locations

Ensure: Optimized vehicle routing policy π_θ

```
Initialize P^4O Environment with the given inputs
Initialize P^4O Agent with an actor-critic network
for episode = 1 to K do
    Reset environment parameters
    R_episode ← 0
    while not all shifts completed do
        for all vehicle v ∈ V do
            Observe current state s_v
            Sample next location a_v ~ π_θ(s_v)
            if an emergency event occurs with probability P^emergency then
                Identify emergency location l_e
                for all vehicle v ∈ V do
                    if v has remaining shift time then
                        last_location ← last visited location of v (or starting point if none)
                        travel_time ← W(last_location, l_e)
                        Subtract travel_time from v's remaining shift time
                        if remaining shift time ≤ 0 then
                            Move v to the next shift and subtract the rest period T^rest
                        end if
                        Append l_e to v's route
                    end if
                end for
                continue to the next decision step
            end if
            Execute action a_v subject to all operational constraints
            if revisit time constraint is violated then
                Apply penalty to the reward
            end if
            Observe reward r_v
            R_episode ← R_episode + r_v
        end for
        Compute returns G_t using discounted rewards
        Estimate advantages A_t
        Update policy π_θ using the clipped objective
    end while
end for
return π_θ
```

convergence while maintaining computational efficiency (Line 3). At the beginning of each episode, the environment parameters are reset in Line 4, and the cumulative reward $\mathcal{R}_{episode}$ is set to zero (Line 5).

In each episode, the algorithm goes through decision steps until all shifts are completed, as indicated in Line 6. For every vehicle $v \in \mathcal{V}$ (Line 7), the current state s_v is observed in Line 8, and the next location a_v is sampled from the policy $\pi_\theta(s_v)$ in Line 9. If an emergency event occurs with a probability of $\mathcal{P}^{emergency}$ (Line 10), an emergency location l_e is identified in Line 11, and a replanning procedure is initiated for all vehicles (Lines 12-19). Specifically, for each vehicle that has remaining shift time (Line 13), the algorithm determines its last visited location (or starting point if none exists) in Line 14, calculates the travel time to l_e using $\mathcal{W}(last_location, l_e)$ in Line 15, and deducts this travel time from the vehicle's remaining shift time (Line 16). If the remaining shift time becomes non-positive, the vehicle moves to the next shift with the rest period $\mathcal{T}^{rest}$ subtracted, and l_e is added to its route (Lines 17 and 18). When a vehicle's remaining shift time is exhausted while responding to an emergency, it immediately transitions to the next shift after deducting the mandatory rest period, thereby upholding operational constraints. Simultaneously, the emergency location is appended to its route, ensuring that urgent incidents are addressed without compromising regulatory requirements. This transition indicates that the current shift has been fully utilized and requires the vehicle to take its necessary rest before resuming operations. The algorithm then proceeds to the next decision step in Line 20. In the absence of an emergency, each vehicle's next location is determined by sampling from the policy $\pi_\theta(s_v)$, where s_v represents the current state and the policy network produces a probability distribution over all candidate locations. Once the action a_v is selected, the vehicle proceeds by updating its route, deducting the corresponding travel and on-site service times from its remaining shift duration, while strictly enforcing operational constraints such as the minimum revisit interval (with penalties for violations) and the mandatory rest period between shifts, which prompts a shift transition when the available time is exhausted (Line 21). The algorithm checks if the revisit time constraint is violated, and if it is, a penalty is applied to the reward in Lines 22 and 23 (the penalty for this violation is -50). This reward structure encourages the agent to avoid significant penalties [4]. The immediate reward r_v, which accounts for both positive outcomes and penalties incurred, is observed and added to $\mathcal{R}_{episode}$ (Lines 23 and 24).

After processing the vehicles at the decision step, the agent calculates the returns $\mathcal{G}_t$ using discounted rewards ($\mathcal{R}_{episode}$) and estimates the advantages $\mathcal{A}_t$, which measure the relative benefit of the selected actions. In this context, $\mathcal{G}_t$ signifies the cumulative sum of discounted future rewards from the current time step onward, reflecting the expected long-term return (Lines 26 and 27). The policy is then updated using the PPO clipped

Table 5.5: $\mathcal{P}^4\mathcal{O}$ Simulation Parameters.

Parameters and Description	**Values**
Activation functions	ReLU
Optimizer	Adam
clip_param (to limit policy updates)	0.18
γ (discount factor)	0.97
λ (bias-variance trade-off)	0.95
α (learning rate)	5e-4
$\mathcal{K}$ (number of episodes)	5000
max_steps (maximum number of steps)	500
hidden_size (neurons in hidden layers)	128
Number of hidden layers	4
update_epochs (implicit, epochs for updates)	4

objective, a crucial aspect of the PPO algorithm, which restricts the policy update by clipping the ratio of new to old action probabilities. The combination of $\mathcal{G}_t$ and $\mathcal{A}_t$ is utilized to compute a surrogate loss that is maximized during the update process, promoting stable and conservative learning as noted in Line 28 [216]. Ultimately, after $\mathcal{K}$ episodes, the algorithm produces the optimized routing policy π_θ, which effectively manages revisit constraints, emergency situations, and overall route coverage.

We chose PPO because it effectively addresses the dynamic challenges of the VPS problem, striking a balance between stability and adaptability in environments with fluctuating routes and unexpected emergencies. Its capability to handle both continuous and discrete states and actions, along with neural network-based policy optimization and a clipping mechanism for stable training, makes it particularly well-suited for this task. PPO is notable for its efficiency in learning from limited interactions while maintaining robustness across various strategic scenarios and hyperparameter settings, requiring minimal tuning [216].

5.4.4 Performance Evaluation

In this section, we compare the performance of the proposed methods, $\mathcal{P}^4\mathcal{O}$ and AHBPS2, across a variety of scenarios, including tests on large-scale graphs representing complex urban networks. The comparison is carried out under different scenarios, using varying fleet sizes, represented by the number of vehicles ($V = 10, 15, 20, 25$), and different scales of the patrol environment, defined by the number of locations ($L = 500, 750, 1000$). In

Figure 5.15: Total visits comparison of $\mathcal{P}^4\mathcal{O}$ & AHBPS2 methods under the mentioned scenarios

the scenarios, V represents vehicles and L represents locations. Our evaluation focuses on dynamic environments, assessing their performance regarding total visits and emergency handling success rate.

The graphs used for testing were generated using real-world maps extracted from Open Source Maps (OSM). By assigning varying numbers of nodes to represent target locations (e.g., banks) on the same map size, we created diverse graph instances with different configurations. This allowed us to simulate a variety of real-world scenarios to test the three approaches comprehensively. Using the NetworkX and OSMNX libraries in Python [309, 310], we processed the extracted maps and converted them into graph structures to serve as inputs for the algorithms. An example of one such generated graph is shown in Fig. 5.4, which corresponds to the map displayed in Fig. 5.3. We use Python3 to simulate the operation of our algorithms. We run our program on AMD Ryzen 7 7700X, Cores 8, and 16 Threads, amplified by NVIDIA GeForce RTX 4070 Super GPU, and 64.0 GB RAM.

Figure 5.16: Training progress for $\mathcal{P}^4\mathcal{O}$

The travel times between nodes are estimated based on distances extracted from the generated maps. A normal distribution with a mean of 15 minutes and a standard deviation of 5 minutes is considered to reflect typical urban travel time variability, which was also observed with most of the examined maps. To introduce realistic fluctuations in traffic conditions, we apply a stochastic adjustment of ± 2 minutes in 25% of the runs, ensuring that the model accounts for varying traffic densities without introducing excessive unpredictability. This accounts for scenarios like low traffic (reduced travel time) or unforeseen incidents (increased travel time). This methodology effectively mitigates randomness, providing a clearer, more accurate reflection of algorithm performance across varying conditions and underscoring the efficiency of the methods. Initially, we fix certain parameters while testing various instances, and after analyzing the results, we adjust inputs to assess their impact. Unless specified otherwise, the rest time is set to 10 minutes, the patrol time per location is 5 minutes, and the travel time distribution maintains a mean of 15 minutes. The results are averaged over 1000 runs for AHBPS2 and over ten runs for $\mathcal{P}^4\mathcal{O}$.

Figure 5.17: Comparison of the proposed methods regarding the total traveled distance (KM)

Fig. 5.15 shows the performance of the proposed methods over the mentioned scenarios. As illustrated, $\mathcal{P}^4\mathcal{O}$ consistently outperforms AHBPS2 by great differences, as expected in all scenarios. This is because $\mathcal{P}^4\mathcal{O}$ can adaptively learn and optimize through trial and error in a way that AHBPS2, which relies on predefined rules, cannot. $\mathcal{P}^4\mathcal{O}$ proves its efficiency over all sorts of networks. For instance, in the scenario with 25V, 750L, $\mathcal{P}^4\mathcal{O}$ covered 689 locations safely out of 750, which is around 92% coverage, and can be utilized and adjusted by companies based on their constraints to meet their demand. These results emphasize how adaptive, data-driven methods capable of iterative learning are better equipped to exploit the added flexibility presented by more extensive and variable networks. Consequently, these findings further confirm the suitability of $\mathcal{P}^4\mathcal{O}$ for real-world applications, where operational scales and coverage requirements are likely to expand over time.

Figure 5.18: Total visits comparison for the proposed methods with different Patrol Time

Fig. 5.16 shows the average reward received by the $\mathcal{P}^4\mathcal{O}$'s agent throughout 3000 episodes. As can be noticed, initially, due to exploration, the agent incurs large negative rewards; however, it eventually discovers an optimal patrolling plan and the reward stabilizes after approximately 200 episodes. The shaded area around the reward shows the standard deviation. As episodes progress, the uncertainty (standard deviation) tends to diminish.

Fig. 5.17 presents the total traveled distance (in KM) across different scenarios. As shown, an increase in the number of vehicles or locations results in greater travel distances. This trend arises for two reasons: first, a larger fleet size leads to more vehicles actively patrolling, thereby increasing the total distance covered; second, a higher number of locations expands the patrol area, providing more destinations for vehicles to visit. Consequently, the overall traveled distance increases as both variables grow. The trends of each algorithm over the scenarios are also shown with different colors. For instance, in the scenario of 20V, 1000L, vehicles traveled around 850 km in total across the whole operation time, 12 hours, for $\mathcal{P}^4\mathcal{O}$. On the other hand, AHBPS2 traveled around 615 km. It can be explained because $\mathcal{P}^4\mathcal{O}$ covered more locations and, as a result, vehicles traveled

Figure 5.19: Total visits comparison for the proposed methods with different Rest Time

more to cover these locations.

We also analyzed how different parameters affect the objective of maximizing visited locations. We applied $\mathcal{P}^4\mathcal{O}$ to large networks with 25 vehicles and 750 locations, as this scenario achieved the highest coverage of 92% among all examined cases. Specifically, we examined the effects of variations in $\mathcal{T}^{rest}$, $\mathcal{T}^{patrol}$, $\mathcal{P}^{emergency}$, and $\mathcal{W}_{l,l'}$ (road conditions). First, we compare different patrol durations of 10 and 15 minutes against the baseline value of 5 minutes. As expected, longer patrol times result in fewer visited locations, as vehicles spend more time at each location (see Fig. 5.18).

Figure 5.20: Performance comparison (total visits) between the proposed methods with different travel time distributions (standard deviation is fixed to 5)

Figure 5.21: Total visits comparison for the proposed methods with different Operation Time

Next, we assessed the impact of varying rest periods at the depot by adjusting the initial 10 minutes to 5 and 15 minutes (Fig. 5.19). Reducing the rest time to 5 minutes yields a slight improvement, while increasing it to 15 minutes diminishes performance. However, shorter rest periods may have unexamined psychological effects, so adequate rest is recommended.

Figure 5.22: Emergency handling success rate comparison for the proposed methods

Lastly, we test different travel time distributions with varying means while keeping the standard deviation at 5. We expect more locations to be visited when the mean travel time is lower since it takes less time to travel from location l to l'. As shown in Fig. 5.20, shorter travel times indeed lead to increased coverage, which meets our expectations.

We also evaluated the effects of having different operation times. We compared two different values (10, & 8 hours) for operation time and compared the results with the base value, 12 hours. As expected, 12 hours has better coverage when it comes to total visits. This is because the fleet has more time to secure locations. Fig 5.21 illustrates these changes. A similar trend is visible for both methods.

Fig. 5.22 reports the emergency handling coverage (%) for both methods. $\mathcal{P}^4\mathcal{O}$ effec-

Figure 5.23: Emergency handling success rate comparison for the proposed methods with different emergency rates

tively handles 98% of the emergency cases happened during the simulations. On the other hand, AHBPS2 performed well, responding in time to around 83% of the emergencies. To conclude, both methods performed very well in handling emergencies. We also tested our proposed methods with different probabilities of emergencies. During our simulations, the probability of an emergency happening ($\mathcal{P}^{emergency}$) was 5%. Fig 5.23 illustrates the effects of different values (2.5% and 10%) along with the baseline. As can be seen, the proposed methods performed very similarly across different probabilities. In higher probabilities of emergency, which is the case for areas with high crime rates, our methods performed slightly better, showcasing their abilities in handling emergencies. It must be mentioned that a higher probability leads to lower coverage since the patrolling plans change and vehicles must travel to new locations.

In conclusion, $\mathcal{P}^4\mathcal{O}$ proves to be highly effective in all sorts of networks and different fleet sizes and demonstrates efficiency in real-time applications due to its high coverage and effective emergency handling system.

5.5 Conclusion

This chapter has examined the Vehicle Patrol Scheduling (VPS) problem by exploring a range of solution methodologies—from classical optimization and heuristic approaches (such as AHBPS and GDVPS) to a reinforcement learning strategy using Proximal Policy Optimization, denoted as $\mathcal{P}^4\mathcal{O}$. Overall, our findings demonstrate that while heuristic methods can deliver rapid scheduling adjustments with low computational cost, their inherent reliance on rule-based improvements means they are prone to getting trapped in local optima and may not scale as effectively to extremely large or highly complex networks. In contrast, the RL-based approach, $\mathcal{P}^4\mathcal{O}$, excels in long-term optimization by dynamically learning patrol policies from experience; however, this advantage comes at the cost of significant training time and computational resources, as well as sensitivity to hyperparameter tuning and potential instability in highly volatile environments.

In particular, $\mathcal{P}^4\mathcal{O}$ achieved high overall coverage and strong emergency handling across various scenarios, yet its extended convergence period and elevated computational demands may hinder its immediate applicability in time-critical settings. Conversely, while GDVPS improves on the basic hill-climbing strategy of AHBPS by better exploring the solution space through genetic operators, both heuristics are ultimately limited by their lack of global optimality guarantees.

These insights suggest that an integrated, hybrid approach, one that leverages the rapid responsiveness of heuristics alongside the adaptive, long-term decision-making capability of reinforcement learning, could offer the most promising direction for future VPS solutions. Further research should focus on reducing the training overhead and sensitivity of RL models and enhancing the exploratory mechanisms of heuristic methods, thereby better balancing short-term responsiveness with long-term optimality in dynamic urban patrol operations.

5.6 Appendix

In this Appendix, we present the linearized forms of the constraints marked with "*" in Section 5.3.2.

Vehicle Visit Time Constraint (Eq.(9))

Let $Z_{c,l,l',v} \in \{0,1\}$ denote a binary decision variable used for linearization. Consequently, the linearized form of Eq. (9) is expressed as follows:

$$\begin{cases} Z_{c,l,l',v} \leq X_{c,l,v-1}, \\ Z_{c,l,l',v} \leq X_{c,l',v}, \end{cases} \quad \forall c \in C,\ l, l' \in N,\ v = 1..v^{\max} \tag{5.17}$$

$$A_{c,v} = A_{c,v-1} + P_{c,v-1} + \sum_{l \in N} \sum_{l' \in N} W_{l,l'} \cdot Z_{c,l,l',v} \quad \forall c \in C,\ v = 2..v^{\max} \tag{5.18}$$

Rest Time End Constraint (Eq.(10))

Let $Z^{AP}_{c,D,v,s} \geq 0$ represent a real decision variable used for linearization, and let M_1 denote a large value greater than any $A_{c,v} + P_{c,v}$. The linearized form of Eq. (10) is then expressed as follows:

$$\begin{cases} Z^{AP}_{c,D,v,s} \leq M_1 \cdot Y_{c,D,v,s}, \\ Z^{AP}_{c,D,v,s} \leq A_{c,v} + P_{c,v}, \end{cases} \quad \forall c \in C,\ v = 1..v^{\max},\ s = 1..s^{\max} \tag{5.19}$$

$$\sum_{s=1}^{s^{\max}} Z^{AP}_{c,D,v,s} \leq R^{end}_c \qquad \forall c \in C,\ v = 1..v^{\max} \tag{5.20}$$

Consecutive Visits Gap Constraint (Eq.(14))

Let $Z^{gap}_{c,l,v,s} \geq 0$ denote a real decision variable used for linearization, and let M_2 represent a large value greater than any $L_{l,(s-1)} + T^{gap} + T^{patrol}$. Consequently, the linearized form of Eq. (14) is expressed as follows:

$$\begin{cases} Z^{gap}_{c,l,v,s} \leq M_2 \cdot Y_{c,l,v,s}, \\ Z^{gap}_{c,l,v,s} \leq L_{l,(s-1)} + T^{gap} + T^{patrol}, \end{cases} \quad \forall c \in C,\ l \in N,\ v = 1..v^{\max},\ s = 2..s^{\max} \tag{5.21}$$

$$A_{c,v} \geq Z^{gap}_{c,l,v,s} \qquad \forall c \in C,\ l \in N,\ v = 1..v^{\max},\ s = 2..s^{\max} \tag{5.22}$$

Minimum Patrolling Duration Constraint (Eq.(15))

Let $Z^{patrol}_{c,l,v,s} \geq 0$ denote a real decision variable introduced for linearization. Additionally, let M_3 represent a sufficiently large constant, greater than any $T^{patrol} + L_{l,s}$. The linearized form of Eq. (15) is then expressed as follows:

$$\begin{cases} Z^{patrol}_{c,l,v,s} \leq M_3 \cdot Y_{c,l,v,s}, \\ Z^{patrol}_{c,l,v,s} \leq T^{patrol} + L_{l,s}, \end{cases} \quad \forall c \in C,\ l \in N,\ v = 1..v^{\max},\ s = 1..s^{\max} \tag{5.23}$$

$$A_{c,v} + P_{c,v} \geq Z^{patrol}_{c,l,v,s} \quad \forall c \in C,\ l \in N,\ v = 1..v^{\max},\ s = 1..s^{\max} \tag{5.24}$$

Chapter 6

Conclusion and Future Work

This thesis makes several concrete contributions to the field of Reinforcement Learning (RL) and its practical application in dynamic, real-world scheduling and routing problems. First, by thoroughly exploring and extending classical RL algorithms, including both model-based and model-free methods, we have laid a solid theoretical foundation that details how RL approaches can be adapted to handle uncertainty, high-dimensional state spaces, and rapidly changing environments. In particular, the integration of deep learning architectures has enabled a significant improvement in scalability and robustness, as evidenced by our analyses of Deep Q-Networks (DQN), Proximal Policy Optimization (PPO), and Actor-Critic methods.

Second, the thesis advances the practical deployment of RL through two in-depth application domains. For Electric Vehicle (EV) routing and charging, our work introduces the TERC Q-learning method (TQL) and a variant of deep Q-learning (DQL) that leverage domain-specific knowledge to dramatically reduce travel distances and charging overhead. These methods were shown to scale effectively to large transportation networks, offering a promising pathway to improve operational efficiency in smart transportation systems.

Third, in the realm of Vehicle Patrol Scheduling (VPS), the proposed RL framework, Patrol Planning with Proximal Policy Optimization ($\mathcal{P}^4\mathcal{O}$), demonstrates considerable advantages over traditional heuristic and optimization approaches. $\mathcal{P}^4\mathcal{O}$ not only achieves superior patrol coverage and faster emergency response but also adapts online to variable demands and uncertainties inherent in urban security operations. In addition, the comparative analysis with heuristic methods such as Adaptive Hill-Climbing-Based Patrol Scheduling (AHBPS) and Genetic-Based Dynamic Vehicle Patrol Scheduling (GDVPS) provides clear evidence of the benefits of integrating learning-based strategies with classical heuristics.

A further contribution of this work is the systematic sensitivity analysis of key operational parameters, such as discount factors, exploration strategies, patrol durations, and rest periods. These analyses yield actionable insights and guidelines that can be directly applied to optimize decision-making systems in both transportation and security contexts.

Despite these advancements, certain limitations remain. The RL models, particularly $\mathcal{P}^4\mathcal{O}$, require significant training time and computational resources, and their performance can be sensitive to hyperparameter tuning. Moreover, while heuristic methods offer rapid response capabilities, they may be prone to local optima and may not always guarantee global optimality in highly complex scenarios. Future research should focus on developing hybrid methods that balance the fast responsiveness of heuristics with the adaptive, long-term learning capabilities of RL.

In conclusion, this thesis bridges the gap between theoretical RL and its practical implementations, demonstrating that thoughtfully designed RL frameworks can significantly outperform conventional approaches in dynamic and high-stakes environments. The insights gained not only contribute to academic research but also pave the way for real-world applications in fleet management, urban transportation, and security surveillance. As advancements in deep learning, optimization techniques, and multi-agent systems continue to emerge, the scalability, adaptability, and efficiency of RL-based frameworks are expected to improve further, expanding their impact on complex, real-world decision-making systems.

References

[1] Majid Ghasemi, Amir Hossein Moosavi, Ibrahim Sorkhoh, Anjali Agrawal, Fadi Alzhouri, and Dariush Ebrahimi. An introduction to reinforcement learning: Fundamental concepts and practical applications. *arXiv preprint arXiv:2408.07712*, 2024.

[2] Hendrik Baier and Peter D Drake. The power of forgetting: Improving the last-good-reply policy in monte carlo go. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):303–309, 2010.

[3] Richard S Sutton. Reinforcement learning: An introduction. *A Bradford Book*, 2018.

[4] Jonas Eschmann. Reward function design in reinforcement learning. *Reinforcement Learning Algorithms: Analysis and Applications*, pages 25–33, 2021.

[5] Richard S Sutton, Andrew G Barto, et al. Reinforcement learning. *Journal of Cognitive Neuroscience*, 11(1):126–134, 1999.

[6] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[7] Dylan J Foster and Alexander Rakhlin. Foundations of reinforcement learning and interactive decision making. *arXiv preprint arXiv:2312.16730*, 2023.

[8] Martin L. Puterman. Chapter 8 markov decision processes. In *Stochastic Models*, volume 2 of *Handbooks in Operations Research and Management Science*, pages 331–434. Elsevier, 1990.

[9] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. In *PhD thesis, King's College, Cambridge*, 1989.

[10] Gavin Anthony Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. Technical report, University of Cambridge, Department of Engineering Cambridge, 1994.

[11] Marco A Wiering and Martijn Van Otterlo. Reinforcement learning. *Adaptation, learning, and optimization*, 12(3):729, 2012.

[12] Damien Ernst and Arthur Louette. Introduction to reinforcement learning, 2024.

[13] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[14] Zihan Ding, Yanhua Huang, Hang Yuan, and Hao Dong. Introduction to reinforcement learning. *Deep reinforcement learning: fundamentals, research and applications*, pages 47–123, 2020.

[15] Roger McFarlane. A survey of exploration strategies in reinforcement learning. *McGill University*, 3:17–18, 2018.

[16] Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, et al. Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038*, 2020.

[17] William F Whitney, Michael Bloesch, Jost Tobias Springenberg, Abbas Abdolmaleki, Kyunghyun Cho, and Martin Riedmiller. Decoupled exploration and exploitation policies for sample-efficient reinforcement learning. *arXiv preprint arXiv:2101.09458*, 2021.

[18] Olivier Caelen and Gianluca Bontempi. A dynamic programming strategy to balance exploration and exploitation in the bandit problem. *Annals of Mathematics and Artificial Intelligence*, 60:3–24, 2010.

[19] Fanghui Liu, Luca Viano, and Volkan Cevher. Understanding deep neural function approximation in reinforcement learning via epsilon-greedy exploration. *Advances in Neural Information Processing Systems*, 35:5093–5108, 2022.

[20] Michael Wunder, Michael L Littman, and Monica Babes. Classes of multiagent q-learning dynamics with epsilon-greedy exploration. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 1167–1174, 2010.

[21] Nicolò Cesa-Bianchi, Claudio Gentile, Gábor Lugosi, and Gergely Neu. Boltzmann exploration done right. *Advances in neural information processing systems*, 30, 2017.

[22] Shaochen Wang, Yuan Pu, Shangtong Yang, Xin Yao, and Bin Li. Boltzmann exploration for deterministic policy optimization. In *Neural Information Processing: 27th International Conference, ICONIP 2020, Bangkok, Thailand, November 23–27, 2020, Proceedings, Part II 27*, pages 214–222. Springer, 2020.

[23] Muhammad Usama and Dong Eui Chang. Learning-driven exploration for reinforcement learning. In *2021 21st International Conference on Control, Automation and Systems (ICCAS)*, pages 1146–1151. IEEE, 2021.

[24] Michael Kearns and Satinder Singh. Near-optimal performance for reinforcement learning in polynomial time. *URL: http://www. research. att. com/˜ mkearns*, 1998.

[25] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990.

[26] Shin Ishii, Wako Yoshida, and Junichiro Yoshimoto. Control of exploitation–exploration meta-parameter in reinforcement learning. *Neural networks*, 15(4-6):665–687, 2002.

[27] Lukas Schäfer, Filippos Christianos, Josiah Hanna, and Stefano V Albrecht. Decoupling exploration and exploitation in reinforcement learning. In *ICML 2021 Workshop on Unsupervised Reinforcement Learning*, 2021.

[28] Haoran Wang, Thaleia Zariphopoulou, and Xunyu Zhou. Exploration versus exploitation in reinforcement learning: A stochastic control approach. *arXiv preprint arXiv:1812.01552*, 2018.

[29] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.

[30] Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

[31] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. *Advances in neural information processing systems*, 29, 2016.

[32] Che Wang, Shuhan Yuan, Kai Shao, and Keith Ross. On the convergence of the monte carlo exploring starts algorithm for reinforcement learning. *arXiv preprint arXiv:2002.03585*, 2020.

[33] James F Peters and Christopher Henry. Approximation spaces in off-policy monte carlo learning. *Engineering applications of artificial intelligence*, 20(5):667–675, 2007.

[34] James F Peters, Daniel Lockery, and Sheela Ramanna. Monte carlo off-policy reinforcement learning: A rough set approach. In *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*, pages 6–pp. IEEE, 2005.

[35] Surya T Tokdar and Robert E Kass. Importance sampling: a review. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(1):54–60, 2010.

[36] Wentao Liu, Tao Tang, Shuai Su, Yuan Cao, Feng Bao, and Jianguo Gao. An intelligent train control approach based on the monte carlo reinforcement learning algorithm. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 1944–1949. IEEE, 2018.

[37] Jayakumar Subramanian and Aditya Mahajan. Renewal monte carlo: Renewal theory-based reinforcement learning. *IEEE Transactions on Automatic Control*, 65(8):3663–3670, 2019.

[38] Vahid Tavakol Aghaei, Arda Ağababaoğlu, Sinan Yıldırım, and Ahmet Onat. A real-world application of markov chain monte carlo method for bayesian trajectory control of a robotic manipulator. *ISA transactions*, 125:580–590, 2022.

[39] Yi Wang, Kok Sung Won, David Hsu, and Wee Sun Lee. Monte carlo bayesian reinforcement learning. *arXiv preprint arXiv:1206.6449*, 2012.

[40] Bo Wu and Yanpeng Feng. Monte-carlo bayesian reinforcement learning using a compact factored representation. In *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*, pages 466–469. IEEE, 2017.

[41] Omer Baykal and Ferda Nur Alpaslan. Reinforcement learning in card game environments using monte carlo methods and artificial neural networks. In *2019 4th International Conference on Computer Science and Engineering (UBMK)*, pages 1–6. IEEE, 2019.

[42] David Siegmund. Importance sampling in the monte carlo study of sequential tests. *The Annals of Statistics*, pages 673–684, 1976.

[43] Supratim Saha and Steven M Kay. Maximum likelihood parameter estimation of superimposed chirps using monte carlo importance sampling. *IEEE Transactions on Signal Processing*, 50(2):224–230, 2002.

[44] Pasi Lassila, Jouni Karvo, and J Virtamo. Efficient importance sampling for monte carlo simulation of multicast networks. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 1, pages 432–439. IEEE, 2001.

[45] Maurice de Koning, Wei Cai, Babak Sadigh, Tomas Oppelstrup, Malvin H Kalos, and Vasily V Bulatov. Adaptive importance sampling monte carlo simulation of rare transition events. *The Journal of chemical physics*, 122(7), 2005.

[46] Stéphane Bulteau and Mohamed El Khadiri. A new importance sampling monte carlo method for a flow network reliability problem. *Naval Research Logistics (NRL)*, 49(2):204–228, 2002.

[47] Abdulrahman Altahhan. Td (0)-replay: An efficient model-free planning with full replay. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2018.

[48] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

[49] Jonathan Baxter, Andrew Tridgell, and Lex Weaver. Knightcap: a chess program that learns by combining td (lambda) with game-tree search. *arXiv preprint cs/9901002*, 1999.

[50] Peter Dayan. The convergence of td (λ) for general λ. *Machine learning*, 8:341–362, 1992.

[51] Peter Dayan and Terrence J Sejnowski. Td (λ) converges with probability 1. *Machine Learning*, 14:295–301, 1994.

[52] Marco A Wiering and Hado Van Hasselt. Two novel on-policy reinforcement learning algorithms based on td (λ)-methods. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 280–287. IEEE, 2007.

[53] Kristopher De Asis and Richard S Sutton. Per-decision multi-step temporal difference learning with control variates. *arXiv preprint arXiv:1807.01830*, 2018.

[54] Kristopher De Asis, J Hernandez-Garcia, G Holland, and Richard Sutton. Multi-step reinforcement learning: A unifying algorithm. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[55] Richard S Sutton and Brian Tanner. Temporal-difference networks. *Advances in neural information processing systems*, 17, 2004.

[56] Zaiwei Chen, Siva Theja Maguluri, Sanjay Shakkottai, and Karthikeyan Shanmugam. A lyapunov theory for finite-sample guarantees of asynchronous q-learning and td-learning variants. *arXiv preprint arXiv:2102.01567*, 2021.

[57] Donghwan Lee. Analysis of off-policy multi-step td-learning with linear function approximation. *arXiv preprint arXiv:2402.15781*, 2024.

[58] Kristopher De Asis. A unified view of multi-step temporal difference learning. 2018.

[59] Janis Zuters. Realizing undelayed n-step td prediction with neural networks. In *Melecon 2010-2010 15th IEEE Mediterranean Electrotechnical Conference*, pages 102–106. IEEE, 2010.

[60] Csaba Szepesvári. *Algorithms for reinforcement learning*. Springer nature, 2022.

[61] Yuhui Wang and Xiaoyang Tan. Greedy multi-step off-policy reinforcement learning. 2020.

[62] Ashique Rupam Mahmood, Huizhen Yu, and Richard S Sutton. Multi-step off-policy learning without importance sampling ratios. *arXiv preprint arXiv:1702.03006*, 2017.

[63] Lisheng Wu and Ke Chen. Bias resilient multi-step off-policy goal-conditioned reinforcement learning. *arXiv preprint arXiv:2311.17565*, 2023.

[64] Yaxin Bi, Adam Thomas-Mitchell, Wei Zhai, and Naveed Khan. A comparative study of deterministic and stochastic policies for q-learning. In *2023 4th International Conference on Artificial Intelligence, Robotics and Control (AIRC)*, pages 1–5. IEEE, 2023.

[65] Sergio Spano, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Matta, Alberto Nannarelli, and Marco Re. An efficient hardware implementation of reinforcement learning: The q-learning algorithm. *Ieee Access*, 7:186340–186351, 2019.

[66] M Ayeesha Nasreen and Selvi Ravindran. An overview of q-learning based energy efficient power allocation in wban (q-eepa). In *2022 2nd International Conference on Innovative Sustainable Computational Technologies (CISCT)*, pages 1–5. IEEE, 2022.

[67] Demei Huang, Haoyuan Zhu, Xianqi Lin, and Li Wang. Application of massive parallel computation based q-learning in system control. In *2022 5th International Conference on Pattern Recognition and Artificial Intelligence (PRAI)*, pages 1–5. IEEE, 2022.

[68] Narayana Darapaneni, Amitavo Basu, Sanket Savla, Raamanathan Gururajan, Najmus Saquib, Sudarshan Singhavi, Aishwarya Kale, Pratik Bid, and Anwesh Reddy Paduri. Automated portfolio rebalancing using q-learning. In *2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pages 0596–0602. IEEE, 2020.

[69] Mayank Daswani, Peter Sunehag, and Marcus Hutter. Q-learning for history-based reinforcement learning. In *Asian Conference on Machine Learning*, pages 213–228. PMLR, 2013.

[70] Baineng Shou, Hehong Zhang, Zhiqiang Long, Yunde Xie, Ke Zhang, and Qiuming Gu. Design and applications of q-learning adaptive pid algorithm for maglev train levitation control system. In *2023 35th Chinese Control and Decision Conference (CCDC)*, pages 1947–1953. IEEE, 2023.

[71] Garika Akshay, Nenavath Srinivas Naik, and Jai Vardhan. Enhancing hadoop performance with q-learning for optimal parameter tuning. In *TENCON 2023-2023 IEEE Region 10 Conference (TENCON)*, pages 617–622. IEEE, 2023.

[72] Mengjiao Yao, Xiang Gao, Jiaxin Wang, and Min Wang. Improving nuclei segmentation in pathological image via reinforcement learning. In *2022 International Conference on Machine Learning, Cloud Computing and Intelligent Mining (MLCCIM)*, pages 290–295. IEEE, 2022.

[73] Chi Jin, Zeyuan Allen-Zhu, Sebastien Bubeck, and Michael I Jordan. Is q-learning provably efficient? *Advances in neural information processing systems*, 31, 2018.

[74] Lucileide MD Da Silva, Matheus F Torquato, and Marcelo AC Fernandes. Parallel implementation of reinforcement learning q-learning technique for fpga. *IEEE Access*, 7:2782–2798, 2018.

[75] Siling Wang and Li Zhang. Q-learning based handover algorithm for high-speed rail wireless communications. In *2023 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2023.

[76] Shufei Wang. Taxi scheduling research based on q-learning. In *2021 3rd International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI)*, pages 700–703. IEEE, 2021.

[77] Wei Xiao, Junyan Chen, Xinmei Li, Min Wang, Danli Huang, and Dongying Zhang. Random walk routing algorithm based on q-learning in optical transport network. In *2022 18th International Conference on Computational Intelligence and Security (CIS)*, pages 88–92. IEEE, 2022.

[78] Xinyu Qu and Minghai Yao. Visual novelty based internally motivated q-learning for mobile robot scene learning and recognition. In *2011 4th International Congress on Image and Signal Processing*, volume 3, pages 1461–1466. IEEE, 2011.

[79] Tereza Paterova, Michal Prauzek, and Jaromir Konecny. Robustness analysis of data-driven self-learning controllers for iot environmental monitoring nodes based on q-learning approaches. In *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 721–727. IEEE, 2022.

[80] Almuthanna Nassar and Yasin Yilmaz. Reinforcement learning for adaptive resource allocation in fog ran for iot with heterogeneous latency requirements. *IEEE Access*, 7:128014–128025, 2019.

[81] Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23, 2010.

[82] Zhizhou Ren, Guangxiang Zhu, Hao Hu, Beining Han, Jianglun Chen, and Chongjie Zhang. On the estimation bias in double q-learning. *Advances in Neural Information Processing Systems*, 34:10246–10259, 2021.

[83] Marouane Ben-Akka, Camel Tanougast, Camille Diou, and Ahmad Chaddad. An efficient hardware implementation of the double q-learning algorithm. In *2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, pages 1–6. IEEE, 2023.

[84] Faezeh Jamshidi, Lei Zhang, and Fahimeh Nezhadalinaei. Autonomous driving systems: Developing an approach based on a* and double q-learning. In *2021 7th International Conference on Web Research (ICWR)*, pages 82–85. IEEE, 2021.

[85] Tereza Paterova, Michal Prauzek, and Jaromir Konecny. Data-driven self-learning controller design approach for power-aware iot devices based on double q-learning strategy. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 01–07. IEEE, 2021.

[86] Jingxin Fan, Tingting Yang, Jian Zhao, Zhengqi Cui, Jiahong Ning, and Pengrui Wang. Double q learning multi-agent routing method for maritime search and rescue. In *2023 International Conference on Ubiquitous Communication (Ucom)*, pages 367–372. IEEE, 2023.

[87] Jaromir Konecny, Michal Prauzek, and Tereza Paterova. Double q-learning adaptive wavelet compression method for data transmission at environmental monitoring stations. In *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 567–572. IEEE, 2022.

[88] Hui Huang, Man Lin, and Qingchen Zhang. Double-q learning-based dvfs for multicore real-time systems. In *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 522–529. IEEE, 2017.

[89] Clark Kendrick Go, Bryan Lao, Junichiro Yoshimoto, and Kazushi Ikeda. A reinforcement learning approach to the shepherding task using sarsa. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3833–3836. IEEE, 2016.

[90] Stefan Wender and Ian Watson. Applying reinforcement learning to small scale combat in the real-time strategy game starcraft: Broodwar. In *2012 ieee conference on computational intelligence and games (cig)*, pages 402–408. IEEE, 2012.

[91] Ankita Jaiswal, Sushil Kumar, and Upasana Dohare. Green computing in heterogeneous internet of things: Optimizing energy allocation using sarsa-based reinforcement learning. In *2020 IEEE 17th India Council International Conference (INDICON)*, pages 1–6. IEEE, 2020.

[92] Dan Wang, Bo Liu, Hongjie Jia, Ziyang Zhang, Jingcheng Chen, and Deyu Huang. Peer-to-peer electricity transaction decisions of the user-side smart energy system based on the sarsa reinforcement learning. *CSEE Journal of Power and Energy Systems*, 8(3):826–837, 2020.

[93] Jihoon Suh and Takashi Tanaka. Sarsa (0) reinforcement learning over fully homomorphic encryption. In *2021 SICE International Symposium on Control Systems (SICE ISCS)*, pages 1–7. IEEE, 2021.

[94] Hitoshi Iima and Yasuaki Kuroe. Swarm reinforcement learning algorithms based on sarsa method. In *2008 SICE Annual Conference*, pages 2045–2049. IEEE, 2008.

[95] Tawfiq M Aljohani and Osama Mohammed. A real-time energy consumption minimization framework for electric vehicles routing optimization based on sarsa reinforcement learning. *Vehicles*, 4(4):1176–1194, 2022.

[96] Harm Van Seijen, Hado Van Hasselt, Shimon Whiteson, and Marco Wiering. A theoretical and empirical analysis of expected sarsa. In *2009 ieee symposium on adaptive dynamic programming and reinforcement learning*, pages 177–184. IEEE, 2009.

[97] Hoomaan Moradimaryamnegari, Marco Frego, and Angelika Peer. Model predictive control-based reinforcement learning using expected sarsa. *IEEE Access*, 10:81177–81191, 2022.

[98] Ivonne Andrea Mantilla Gonzalez and Volker Turau. Comparison of wifi interference mitigation strategies in dsme networks: Leveraging reinforcement learning with expected sarsa. In *2023 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, pages 270–275. IEEE, 2023.

[99] Rasananda Muduli, Debashisha Jena, and Tukaram Moger. Application of expected sarsa-learning for load frequency control of multi-area power system. In *2023 5th International Conference on Energy, Power and Environment: Towards Flexible Green Energy Technologies (ICEPE)*, pages 1–6. IEEE, 2023.

[100] Eunsung Oh and Hanho Wang. Reinforcement-learning-based energy storage system operation strategies to manage wind power forecast uncertainty. *IEEE Access*, 8:20965–20976, 2020.

[101] Akshay Kekuda, R Anirudh, and Mithun Krishnan. Reinforcement learning based intelligent traffic signal control using n-step sarsa. In *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, pages 379–384. IEEE, 2021.

[102] Vivek Kuchibhotla, P Harshitha, and Shobhit Goyal. An n-step look ahead algorithm using mixed (on and off) policy reinforcement learning. In *2020 3rd International Conference on Intelligent Sustainable Systems (ICISS)*, pages 677–681. IEEE, 2020.

[103] Eduardo F Morales and Julio H Zaragoza. An introduction to reinforcement learning. In *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions*, pages 63–80. IGI Global, 2012.

[104] Majid Ghasemi, Amir Hossein Mousavi, and Dariush Ebrahimi. Comprehensive survey of reinforcement learning: From algorithms to practical challenges. *arXiv preprint arXiv:2411.18892*, 2024.

[105] Ian Goodfellow. Deep learning, 2016.

[106] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[107] Geoffrey E Hinton et al. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, volume 1, page 12. Amherst, MA, 1986.

[108] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[109] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[110] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[111] Vaswani Ashish. Attention is all you need. *Advances in neural information processing systems*, 30:I, 2017.

[112] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, page 2. Minneapolis, Minnesota, 2019.

[113] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[114] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[115] John D Kelleher. *Deep learning*. MIT press, 2019.

[116] Nicole Rusk. Deep learning. *Nature Methods*, 13(1):35–35, 2016.

[117] Jorge Ramírez, Wen Yu, and Adolfo Perrusquía. Model-free reinforcement learning from expert demonstrations: a survey. *Artificial Intelligence Review*, 55(4):3213–3241, 2022.

[118] Thomas Degris, Patrick M Pilarski, and Richard S Sutton. Model-free reinforcement learning with continuous action in practice. In *2012 American control conference (ACC)*, pages 2177–2182. IEEE, 2012.

[119] Yongshuai Liu, Avishai Halev, and Xin Liu. Policy learning with constraints in model-free reinforcement learning: A survey. In *The 30th international joint conference on artificial intelligence (ijcai)*, 2021.

[120] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[121] Volodymyr Mnih. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[122] Xu Wang, Sen Wang, Xingxing Liang, Dawei Zhao, Jincai Huang, Xin Xu, Bin Dai, and Qiguang Miao. Deep reinforcement learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(4):5064–5078, 2022.

[123] Ki Uhn Ahn and Cheol Soo Park. Application of deep q-networks for model-free optimal control balancing between different hvac systems. *Science and Technology for the Built Environment*, 26(1):61–74, 2020.

[124] Joseph N Stember and Hrithwik Shalu. Reinforcement learning using deep q networks and q learning accurately localizes brain tumors on mri with very small training sets. *BMC Medical Imaging*, 22(1):224, 2022.

[125] Haobo Guo, Runze Wu, Bing Qi, and Chen Xu. Deep-q-networks-based adaptive dual-mode energy-efficient routing in rechargeable wireless sensor networks. *IEEE Sensors Journal*, 22(10):9956–9966, 2022.

[126] Fatma M Talaat. Effective deep q-networks (edqn) strategy for resource allocation based on optimized reinforcement learning algorithm. *Multimedia Tools and Applications*, 81(28):39945–39961, 2022.

[127] Changhee Song, Heeyun Lee, Kyunghyun Kim, and Suk Won Cha. A power management strategy for parallel phev using deep q-networks. In *2018 IEEE Vehicle Power and Propulsion Conference (VPPC)*, pages 1–5. IEEE, 2018.

[128] Seonghun Yoon and Kyung-Joong Kim. Deep q networks for visual fighting game ai. In *2017 IEEE conference on computational intelligence and games (CIG)*, pages 306–308. IEEE, 2017.

[129] Liangheng Lv, Sunjie Zhang, Derui Ding, and Yongxiong Wang. Path planning via an improved dqn-based learning policy. *IEEE Access*, 7:67319–67330, 2019.

[130] Ahmed S Zamzam, Bo Yang, and Nicholas D Sidiropoulos. Energy storage management via deep q-networks. In *2019 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2019.

[131] Ning Gao, Zhijin Qin, Xiaojun Jing, Qiang Ni, and Shi Jin. Anti-intelligent uav jamming strategy via deep q-networks. *IEEE Transactions on Communications*, 68(1):569–581, 2019.

[132] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[133] Baiyu Peng, Qi Sun, Shengbo Eben Li, Dongsuk Kum, Yuming Yin, Junqing Wei, and Tianyu Gu. End-to-end autonomous driving through dueling double deep q-network. *Automotive Innovation*, 4:328–337, 2021.

[134] Xin Zhang, Xiaoxu Shi, Zuqiong Zhang, Zhengzhong Wang, and Lieping Zhang. A ddqn path planning algorithm based on experience classification and multi steps for mobile robots. *Electronics*, 11(14):2120, 2022.

[135] Amjad Iqbal, Mau-Luen Tham, and Yoong Choon Chang. Double deep q-network-based energy-efficient resource allocation in cloud radio access network. *IEEE Access*, 9:20440–20449, 2021.

[136] Guoyu Zuo, Tingting Du, and Jiahao Lu. Double dqn method for object detection. In *2017 Chinese Automation Congress (CAC)*, pages 6727–6732. IEEE, 2017.

[137] Wenyu Zhang, Jingyao Gai, Zhigang Zhang, Lie Tang, Qingxi Liao, and Youchun Ding. Double-dqn based path smoothing and tracking control method for robotic vehicle navigation. *Computers and Electronics in Agriculture*, 166:104985, 2019.

[138] Shuwei Zhang, Yutian Wu, Harutoshi Ogai, Hiroshi Inujima, and Shigeyuki Tateno. Tactical decision-making for autonomous driving using dueling double deep q network with double attention. *IEEE Access*, 9:151983–151992, 2021.

[139] Rui Li, Wenyin Gong, Ling Wang, Chao Lu, Zixiao Pan, and Xinying Zhuang. Double dqn-based coevolution for green distributed heterogeneous hybrid flowshop scheduling with multiple priorities of jobs. *IEEE Transactions on Automation Science and Engineering*, 2023.

[140] Xidi Xue, Zhan Li, Dongsheng Zhang, and Yingxin Yan. A deep reinforcement learning method for mobile robot collision avoidance based on double dqn. In *2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)*, pages 2131–2136. IEEE, 2019.

[141] Shuojie Mo, Xiaofei Pei, and Zhenfu Chen. Decision-making for oncoming traffic overtaking scenario using double dqn. In *2019 3rd Conference on Vehicle Control and Intelligence (CVCI)*, pages 1–4. IEEE, 2019.

[142] Yue Yu, Yufei Liu, Jichun Wang, Noboru Noguchi, and Yong He. Obstacle avoidance method based on double dqn for agricultural robots. *Computers and Electronics in Agriculture*, 204:107546, 2023.

[143] Yang Xiaofei, Shi Yilun, Liu Wei, Ye Hui, Zhong Weibo, and Xiang Zhengrong. Global path planning algorithm based on double dqn for multi-tasks amphibious unmanned surface vehicle. *Ocean Engineering*, 266:112809, 2022.

[144] Changsung Lee, Jaewook Jung, and Jong-Moon Chung. Intelligent dual active protocol stack handover based on double dqn deep reinforcement learning for 5g mmwave networks. *IEEE Transactions on Vehicular Technology*, 71(7):7572–7584, 2022.

[145] Yi Zhang, Ping Sun, Yuhan Yin, Lin Lin, and Xuesong Wang. Human-like autonomous vehicle speed control by deep reinforcement learning with double q-learning. In *2018 IEEE intelligent vehicles symposium (IV)*, pages 1251–1256. IEEE, 2018.

[146] Dongji Li, Shaoyi Xu, and Jianan Zhao. Partially observable double dqn based iot scheduling for energy harvesting. In *2019 IEEE international conference on communications workshops (ICC Workshops)*, pages 1–6. IEEE, 2019.

[147] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[148] Bao-An Han and Jian-Jun Yang. Research on adaptive job shop scheduling problems based on dueling double dqn. *Ieee Access*, 8:186474–186495, 2020.

[149] Tae-Won Ban. An autonomous transmission scheme using dueling dqn for d2d communication networks. *IEEE transactions on vehicular technology*, 69(12):16348–16352, 2020.

[150] Zhihao Xuan, Guiyi Wei, and Zhengwei Ni. Power allocation in multi-agent networks via dueling dqn approach. In *2021 IEEE 6th International Conference on Signal and Image Processing (ICSIP)*, pages 959–963. IEEE, 2021.

[151] Yu Liu and Cong Zhang. Application of dueling dqn and decga for parameter estimation in variogram models. *IEEE Access*, 8:38112–38122, 2020.

[152] Linqiang Huang, Miao Ye, Xingsi Xue, Yong Wang, Hongbing Qiu, and Xiaofang Deng. Intelligent routing method based on dueling dqn reinforcement learning and network traffic state prediction in sdn. *Wireless Networks*, 30(5):4507–4525, 2024.

[153] Wenqi Liu, Pengbo Si, Enchang Sun, Meng Li, Chao Fang, and Yanhua Zhang. Green mobility management in uav-assisted iot based on dueling dqn. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[154] Seifu Birhanu Tadele, Binayak Kar, Frezer Guteta Wakgra, and Asif Uddin Khan. Optimization of end-to-end aoi in edge-enabled vehicular fog systems: A dueling-dqn approach. *arXiv preprint arXiv:2407.02815*, 2024.

[155] Weilai Jiang, Chong Bao, Guoqiang Xu, and Yaonan Wang. Research on autonomous obstacle avoidance and target tracking of uav based on improved dueling dqn algorithm. In *2021 China Automation Congress (CAC)*, pages 5110–5115. IEEE, 2021.

[156] Zhijian Huang, Sen Liu, and Guichen Zhang. The usv path planning of dueling dqn algorithm based on tree sampling mechanism. In *2022 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC)*, pages 971–976. IEEE, 2022.

[157] David Silver and Joel Veness. Monte-carlo planning in large pomdps. *Advances in neural information processing systems*, 23, 2010.

[158] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[159] Leonardo Rossi, Mark HM Winands, and Christoph Butenweg. Monte carlo tree search as an intelligent search tool in structural design problems. *Engineering with Computers*, 38(4):3219–3236, 2022.

[160] Michael C Fu. A tutorial introduction to monte carlo tree search. In *2020 Winter Simulation Conference (WSC)*, pages 1178–1193. IEEE, 2020.

[161] Weijia Wang and Michele Sebag. Multi-objective monte-carlo tree search. In *Asian conference on machine learning*, pages 507–522. PMLR, 2012.

[162] Xiu Su, Tao Huang, Yanxi Li, Shan You, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Prioritized architecture sampling with monto-carlo tree search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10968–10977, 2021.

[163] Nicholas Zerbel and Logan Yliniemi. Multiagent monte carlo tree search. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2309–2311, 2019.

[164] Diego Perez, Sanaz Mostaghim, Spyridon Samothrakis, and Simon M Lucas. Multi-objective monte carlo tree search for real-time games. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(4):347–360, 2014.

[165] Hendrik Baier and Mark HM Winands. Monte-carlo tree search and minimax hybrids. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pages 1–8. IEEE, 2013.

[166] Maarten De Waard, Diederik M Roijers, and Sander CJ Bakkes. Monte carlo tree search with options for general video game playing. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.

[167] Marc Lanctot, Mark HM Winands, Tom Pepels, and Nathan R Sturtevant. Monte carlo tree search with heuristic evaluations using implicit minimax backups. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014.

[168] Mark HM Winands, Yngvi Bjornsson, and Jahn-Takeshi Saito. Monte carlo tree search in lines of action. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):239–250, 2010.

[169] André Santos, Pedro A Santos, and Francisco S Melo. Monte carlo tree search experiments in hearthstone. In *2017 IEEE conference on computational intelligence and games (CIG)*, pages 272–279. IEEE, 2017.

[170] Peter I Cowling, Edward J Powley, and Daniel Whitehouse. Information set monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):120–143, 2012.

[171] Paolo Ciancarini and Gian Piero Favini. Monte carlo tree search in kriegspiel. *Artificial Intelligence*, 174(11):670–684, 2010.

[172] Thomas M Moerland, Joost Broekens, Aske Plaat, and Catholijn M Jonker. Monte carlo tree search for asymmetric trees. *arXiv preprint arXiv:1805.09218*, 2018.

[173] Andrew W Moore and Christopher G Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 13:103–130, 1993.

[174] Jing Peng and Ronald J Williams. Efficient learning and planning within the dyna framework. *Adaptive behavior*, 1(4):437–454, 1993.

[175] Ruizhi Li, Qing Wang, Chaoyang Dong, et al. Morphing strategy design for uav based on prioritized sweeping reinforcement learning. In *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, pages 2786–2791. IEEE, 2020.

[176] Roman Zajdel. Epoch-incremental dyna-learning and prioritized sweeping algorithms. *Neurocomputing*, 319:13–20, 2018.

[177] Harm Van Seijen and Rich Sutton. Planning by prioritized sweeping with small backups. In *International Conference on Machine Learning*, pages 361–369. PMLR, 2013.

[178] Eugenio Bargiacchi, Timothy Verstraeten, Diederik M Roijers, and Ann Nowé. Model-based multi-agent reinforcement learning with cooperative prioritized sweeping. *arXiv preprint arXiv:2001.07527*, 2020.

[179] Rahul M Desai and BP Patil. Prioritized sweeping reinforcement learning based routing for manets. *Indonesian Journal of Electrical Engineering and Computer Science*, 5(2):383–390, 2017.

[180] Richard Dearden. Structured prioritized sweeping. In *ICML*, pages 82–89. Citeseer, 2001.

[181] Matilde Santos, Victoria López, Guillermo Botella, et al. Dyna-h: A heuristic planning reinforcement learning algorithm applied to role-playing game strategy decision systems. *Knowledge-Based Systems*, 32:28–36, 2012.

[182] Yuan Chai and Xiao-Jun Zeng. A multi-objective dyna-q based routing in wireless mesh network. *Applied Soft Computing*, 108:107486, 2021.

[183] Silvia Del Giorno, Federico D'Antoni, Vincenzo Piemonte, and Mario Merone. A new glycemic closed-loop control based on dyna-q for type-1-diabetes. *Biomedical Signal Processing and Control*, 81:104492, 2023.

[184] Tarek Faycal and Claudio Zito. Dyna-t: Dyna-q and upper confidence bounds applied to trees. *arXiv preprint arXiv:2201.04502*, 2022.

[185] Zhipeng Xu, Fei Zhu, Yuchen Fu, Quan Liu, and Shuhua You. A dyna-q based multi-path load-balancing routing algorithm in wireless sensor networks. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 3, pages 1–4. IEEE, 2015.

[186] Xuesi Li, Chunlei Yang, Jie Song, Shuo Feng, Wenqiao Li, and Hua He. A motion control method for agent based on dyna-q algorithm. In *2023 4th International Conference on Computer Engineering and Application (ICCEA)*, pages 274–278. IEEE, 2023.

[187] Muleilan Pei, Hao An, Bo Liu, and Changhong Wang. An improved dyna-q algorithm for mobile robot path planning in unknown dynamic environment. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(7):4415–4425, 2021.

[188] Fan Wang, Jie Gao, Mushu Li, and Lian Zhao. Autonomous pev charging scheduling using dyna-q reinforcement learning. *IEEE Transactions on Vehicular Technology*, 69(11):12609–12620, 2020.

[189] Yiyang Liu, Shuaihua Yan, Yang Zhao, Chunhe Song, and Fei Li. Improved dyna-q: a reinforcement learning method focused via heuristic graph for agv path planning in dynamic environments. *Drones*, 6(11):365, 2022.

[190] Guoliang Zhang, Yonggui Li, Yingtao Niu, and Quan Zhou. Anti-jamming path selection method in a wireless communication network based on dyna-q. *Electronics*, 11(15):2397, 2022.

[191] Kao-Shing Hwang, Wei-Cheng Jiang, and Yu-Jen Chen. Model learning and knowledge sharing for a multiagent system with dyna-q learning. *IEEE transactions on cybernetics*, 45(5):978–990, 2014.

[192] Jingyi Huang, Qingke Tan, Jiming Ma, and Liang Han. Path planning method using dyna-q algorithm under complex urban environment. In *2022 China Automation Congress (CAC)*, pages 6776–6781. IEEE, 2022.

[193] Emanuele Vitolo, Alberto San Miguel, Javier Civera, and Cristian Mahulea. Performance evaluation of the dyna-q algorithm for robot navigation. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 322–327. IEEE, 2018.

[194] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[195] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[196] Junzi Zhang, Jongho Kim, Brendan O'Donoghue, and Stephen Boyd. Sample efficient reinforcement learning with reinforce. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 10887–10895, 2021.

[197] Liang Guo, Zhongliang Li, Rachid Outbib, and Fei Gao. Function approximation reinforcement learning of energy management with the fuzzy reinforce for fuel cell hybrid electric vehicles. *Energy and AI*, 13:100246, 2023.

[198] Julie C Lauffenburger, Elad Yom-Tov, Punam A Keller, Marie E McDonnell, Lily G Bessette, Constance P Fontanet, Ellen S Sears, Erin Kim, Kaitlin Hanken, J Joseph Buckley, et al. Reinforcement learning to improve non-adherence for diabetes treatments by optimising response and customising engagement (reinforce): study protocol of a pragmatic randomised trial. *BMJ open*, 11(12):e052091, 2021.

[199] Ye Tao and Wee Lum Tan. A reinforcement learning approach to wi-fi rate adaptation using the reinforce algorithm. In *2024 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2024.

[200] Kasun Weerakoon, Souradip Chakraborty, Nare Karapetyan, Adarsh Jagan Sathyamoorthy, Amrit Singh Bedi, and Dinesh Manocha. Htron: Efficient outdoor navigation with sparse rewards via heavy tailed adaptive reinforce algorithm. *arXiv preprint arXiv:2207.03694*, 2022.

[201] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[202] Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 4. Athena scientific, 2012.

[203] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[204] István Szita and András Lörincz. Learning tetris using the noisy cross-entropy method. *Neural computation*, 18(12):2936–2941, 2006.

[205] Boyi Liu, Qi Cai, Zhuoran Yang, and Zhaoran Wang. Neural trust region/proximal policy optimization attains globally optimal policy. *Advances in neural information processing systems*, 32, 2019.

[206] Joshua Achiam. Advanced policy gradient methods, 2017.

[207] Qunyong Yuan and Nanfeng Xiao. A monotonic policy optimization algorithm for high-dimensional continuous control problem in 3d mujoco. *Multimedia Tools and Applications*, 78:28665–28680, 2019.

[208] Sahar Roostaie and Mohammad Mehdi Ebadzadeh. Entrpo: Trust region policy optimization method with entropy regularization. *arXiv preprint arXiv:2110.13373*, 2021.

[209] Jakub Grudzien Kuba, Ruiqing Chen, Muning Wen, Ying Wen, Fanglei Sun, Jun Wang, and Yaodong Yang. Trust region policy optimisation in multi-agent reinforcement learning. *arXiv preprint arXiv:2109.11251*, 2021.

[210] Huaping Liu, Yupei Wu, and Fuchun Sun. Extreme trust region policy optimization for active object recognition. *IEEE transactions on neural networks and learning systems*, 29(6):2253–2258, 2018.

[211] Judhi Santoso et al. Multiagent simulation on hide and seek games using policy gradient trust region policy optimization. In *2020 7th International Conference on Advance Informatics: Concepts, Theory and Applications (ICAICTA)*, pages 1–5. IEEE, 2020.

[212] Biswajit Mondal, Abhijit Banerjee, and Subir Gupta. Xss filter detection using trust region policy optimization. In *2023 1st International Conference on Advanced Innovations in Smart Cities (ICAISC)*, pages 1–4. IEEE, 2023.

[213] Joren Erens. Universal robot policy: Using a surrogate model in combination with trpo. Master's thesis, University of Twente, 2024.

[214] Kuthsav Thattai, Jayashri Ravishankar, and Chaojie Li. Consumer-centric home energy management system using trust region policy optimization-based multi-agent deep reinforcement learning. In *2023 IEEE Belgrade PowerTech*, pages 1–6. IEEE, 2023.

[215] Nuoheng Peng, Yan Lin, Yijin Zhang, and Jun Li. Aoi-aware joint spectrum and power allocation for internet of vehicles: A trust region policy optimization-based approach. *IEEE Internet of Things Journal*, 9(20):19916–19927, 2022.

[216] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[217] Haoran Wei, Xuanzhang Liu, Lena Mashayekhy, and Keith Decker. Mixed-autonomy traffic control with proximal policy optimization. In *2019 IEEE Vehicular Networking Conference (VNC)*, pages 1–8. IEEE, 2019.

[218] Bei Zhang, Xiao Lu, Ruisheng Diao, Haifeng Li, Tu Lan, Di Shi, and Zhiwei Wang. Real-time autonomous line flow control using proximal policy optimization. In *2020 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2020.

[219] Cheng-Shuo Ying, Andy HF Chow, Yi-Hui Wang, and Kwai-Sang Chin. Adaptive metro service schedule and train composition with a proximal policy optimization approach based on deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):6895–6906, 2021.

[220] Jiangliang Jin and Yunjian Xu. Optimal policy characterization enhanced proximal policy optimization for multitask scheduling in cloud computing. *IEEE Internet of Things Journal*, 9(9):6418–6433, 2021.

[221] Le Zhang, Yanshuo Zhang, Xin Zhao, and Zexiao Zou. Image captioning via proximal policy optimization. *Image and Vision Computing*, 108:104126, 2021.

[222] Yang Guan, Yangang Ren, Shengbo Eben Li, Qi Sun, Laiquan Luo, and Keqiang Li. Centralized cooperation for connected and automated vehicles at intersections by proximal policy optimization. *IEEE Transactions on Vehicular Technology*, 69(11):12597–12608, 2020.

[223] Eivind Bøhn, Erlend M Coates, Signe Moe, and Tor Ame Johansen. Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization. In *2019 international conference on unmanned aircraft systems (ICUAS)*, pages 523–533. IEEE, 2019.

[224] Guilherme Cano Lopes, Murillo Ferreira, Alexandre da Silva Simões, and Esther Luna Colombini. Intelligent control of a quadrotor with proximal policy optimization reinforcement learning. In *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*, pages 503–508. IEEE, 2018.

[225] Fei Ye, Xuxin Cheng, Pin Wang, Ching-Yao Chan, and Jiucai Zhang. Automated lane change strategy using proximal policy optimization-based deep reinforcement

learning. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1746–1752. IEEE, 2020.

[226] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, part C (applications and reviews)*, 42(6):1291–1307, 2012.

[227] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[228] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[229] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[230] Yun Zou, Qing-Zi Xing, Bai-Chuan Wang, Shu-Xin Zheng, Cheng Cheng, Zhong-Ming Wang, and Xue-Wu Wang. Application of the asynchronous advantage actor–critic machine learning algorithm to real-time accelerator tuning. *Nuclear Science and Techniques*, 30:1–9, 2019.

[231] Min-Yuh Day, Ching-Ying Yang, and Yensen Ni. Portfolio dynamic trading strategies using deep reinforcement learning. *Soft Computing*, 28(15):8715–8730, 2024.

[232] Sudheer Mangalampalli, Ganesh R Karri, Sachi N Mohanty, Shahid Ali, M Ijaz Khan, Sherzod Abdullaev, and Salman A AlQahtani. Multi-objective prioritized task scheduler using improved asynchronous advantage actor critic (a3c) algorithm in multi cloud environment. *IEEE Access*, 2024.

[233] Fan Jiang, Shaojiang Han, and Changyin Sun. Asynchronous advantage actor-critic algorithm based cooperative caching strategy for fog radio access networks. In *2023 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2023.

[234] Jianbo Du, Wenjie Cheng, Guangyue Lu, Haotong Cao, Xiaoli Chu, Zhicai Zhang, and Junxuan Wang. Resource pricing and allocation in mec enabled blockchain systems: An a3c deep reinforcement learning approach. *IEEE Transactions on Network Science and Engineering*, 9(1):33–44, 2021.

[235] M Joypriyanka and R Surendran. Chess game to improve the mental ability of alzheimer's patients using a3c. In *2023 Fifth International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–6. IEEE, 2023.

[236] Teckchai Tiong, Ismail Saad, Kenneth Tze Kin Teo, and Herwansyah Bin Lago. Autonomous valet parking with asynchronous advantage actor-critic proximal policy optimization. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0334–0340. IEEE, 2022.

[237] Zhuoyang Shi, Lixin Li, Yang Xu, Xu Li, Wei Chen, and Zhu Han. Content caching policy for 5g network based on asynchronous advantage actor-critic method. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2019.

[238] Jing Yang, JiaLin Lu, Xu Zhou, ShaoBo Li, ChuanYue Xiong, and Jianjun Hu. Ha-a2c: Hard attention and advantage actor-critic for addressing latency optimization in edge computing. *IEEE Transactions on Green Communications and Networking*, 2024.

[239] Prashanth Choppara and Sudheer Mangalampalli. Reliability and trust aware task scheduler for cloud-fog computing using advantage actor critic (a2c) algorithm. *IEEE Access*, 2024.

[240] Ycaro Dantas, Pedro Enrique Iturria-Rivera, Hao Zhou, Majid Bavand, Medhat Elsayed, Raimundas Gaigalas, and Melike Erol-Kantarci. Beam selection for energy-efficient mmwave network using advantage actor critic learning. In *ICC 2023-IEEE International Conference on Communications*, pages 5285–5290. IEEE, 2023.

[241] Hyukjoon Kwon. Learning-based power delay profile estimation for 5g nr via advantage actor-critic (a2c). In *2022 IEEE 95th Vehicular Technology Conference:(VTC2022-Spring)*, pages 1–6. IEEE, 2022.

[242] M Vishal, Yatin Satija, and B Sathish Babu. Trading agent for the indian stock market scenario using actor-critic based reinforcement learning. In *2021 IEEE international conference on computation system and information technology for sustainable solutions (CSITSS)*, pages 1–5. IEEE, 2021.

[243] Yukun Sun and Xing Zhang. A2c learning for tasks segmentation with cooperative computing in edge computing networks. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 2236–2241. IEEE, 2022.

[244] Xiao Han, Jing Wang, Qinyu Zhang, Xue Qin, and Meng Sun. Multi-uav automatic dynamic obstacle avoidance with experience-shared a2c. In *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 330–335. IEEE, 2019.

[245] Kun Shao, Dongbin Zhao, Yuanheng Zhu, and Qichao Zhang. Visual navigation with actor-critic deep reinforcement learning. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2018.

[246] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

[247] Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

[248] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[249] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[250] George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

[251] Yu Kong, Ying Li, Jiandong Wang, and Shiwei Yin. Edge computing task unloading decision optimization algorithm based on deep reinforcement learning. In *China Conference on Wireless Sensor Networks*, pages 189–201. Springer, 2023.

[252] Angelo Candeli, Gianmaria De Tommasi, Dario Giuseppe Lui, Adriano Mele, Stefania Santini, and Gaetano Tartaglione. A deep deterministic policy gradient learning approach to missile autopilot design. *IEEE Access*, 10:19685–19696, 2022.

[253] Zhongbao Wei, Zhongyi Quan, Jingda Wu, Yang Li, Josep Pou, and Hao Zhong. Deep deterministic policy gradient-drl enabled multiphysics-constrained fast charging of lithium-ion battery. *IEEE Transactions on Industrial Electronics*, 69(3):2588–2598, 2021.

[254] Shuhuan Wen, Jianhua Chen, Shen Wang, Hong Zhang, and Xueheng Hu. Path planning of humanoid arm based on deep deterministic policy gradient. In *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1755–1760. IEEE, 2018.

[255] Xiaoshan Gao, Liang Yan, Zhijun Li, Gang Wang, and I-Ming Chen. Improved deep deterministic policy gradient for dynamic obstacle avoidance of mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 53(6):3675–3682, 2023.

[256] Shiyang Zhou, Yufan Cheng, Xia Lei, and Huanhuan Duan. Deep deterministic policy gradient with prioritized sampling for power control. *IEEE Access*, 8:194240–194250, 2020.

[257] Yanchang Liang, Chunlin Guo, Zhaohao Ding, and Huichun Hua. Agent-based modeling in electricity market using deep deterministic policy gradient algorithm. *IEEE transactions on power systems*, 35(6):4180–4192, 2020.

[258] Yi-Han Xu, Cheng-Cheng Yang, Min Hua, and Wen Zhou. Deep deterministic policy gradient (ddpg)-based resource allocation scheme for noma vehicular communications. *IEEE Access*, 8:18797–18807, 2020.

[259] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[260] Sebastian B Thrun. *Efficient exploration in reinforcement learning*. Carnegie Mellon University, 1992.

[261] Ozan Yazar, Serdar Coskun, Lin Li, Fengqi Zhang, and Cong Huang. Actor-critic td3-based deep reinforcement learning for energy management strategy of hev. In *2023 5th International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pages 1–6. IEEE, 2023.

[262] Najmaddin Abo Mosali, Syariful Syafiq Shamsudin, Omar Alfandi, Rosli Omar, and Najib Al-Fadhali. Twin delayed deep deterministic policy gradient-based target tracking for unmanned aerial vehicle with achievement rewarding and multistage training. *IEEE Access*, 10:23545–23559, 2022.

[263] Xinyang Han, Zongwang Li, and Zhuochen Xie. Two-step random access optimization for 5g-and-beyond leo satellite communication system: a td3-based msga channel allocation strategy. *IEEE Communications Letters*, 27(6):1570–1574, 2023.

[264] Rakib Hossain, Mukesh Gautam, Mohammad Mansour Lakouraj, Hanif Livani, and Mohammed Benidris. Volt-var optimization in distribution networks using twin delayed deep reinforcement learning. In *2022 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, pages 1–5. IEEE, 2022.

[265] Mazen Shehab, Ahmed Zaghloul, and Ayman El-Badawy. Low-level control of a quadrotor using twin delayed deep deterministic policy gradient (td3). In *2021 18th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, pages 1–6. IEEE, 2021.

[266] Jiaming Yu, Hao Sun, and Qinglin Sun. Multi-auv charging navigation trajectory planning based on twin delayed deep deterministic policy gradient. In *2023 42nd Chinese Control Conference (CCC)*, pages 8521–8526. IEEE, 2023.

[267] Himanshu Kumar Bishen, KV Shihabudheen, and PP Muhammed Shanir. Adaptive cruise control using twin delayed deep deterministic policy gradient. In *2023 5th International Conference on Energy, Power and Environment: Towards Flexible Green Energy Technologies (ICEPE)*, pages 1–6. IEEE, 2023.

[268] Edouard Leurent. An environment for autonomous driving decision-making. https://github.com/eleurent/highway-env, 2018.

[269] Li Zhang, Zhongshan Liu, Lan Yu, Ke Fang, Baozhen Yao, and Bin Yu. Routing optimization of shared autonomous electric vehicles under uncertain travel time and uncertain service time. *Transportation Research Part E: Logistics and Transportation Review*, 157:102548, 2022.

[270] Jennifer Morris, David Hone, Martin Haigh, Andrei Sokolov, and Sergey Paltsev. Future energy: in search of a scenario reflecting current and future pressures and trends. *Environmental Economics and Policy Studies*, 25, 2022.

[271] Mohamed Nadir Boukoberine, Teresa Donateo, and Mohamed Benbouzid. Optimized energy management strategy for hybrid fuel cell powered drones in persistent missions using real flight test data. *IEEE Transactions on Energy Conversion*, 37(3):2080–2091, 2022.

[272] Dariush Ebrahimi, Seyedmohammad Kashefi Mofrad, Maryam Helae, Sabah Mohammed, and Fadi Alzhouri. Fastest route and charging optimization of an electric vehicle with battery's life consideration. In *Proceedings of the Int'l ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications*, pages 39–46, 2023.

[273] Dariush Ebrahimi, Seyedmohammad Kashefi, Thiago E Alves de Oliveira, and Fadi Alzhouri. Efficient routing and charging strategy for electric vehicles considering battery life: A reinforcement learning approach. In *2024 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 429–435. IEEE, 2024.

[274] Shuai Zhang, Yuvraj Gajpal, S.S. Appadoo, and M.M.S. Abdulkader. Electric vehicle routing problem with recharging stations for minimizing energy consumption. *International Journal of Production Economics*, 203:404–413, 2018.

[275] Gerhard Hiermann, Jakob Puchinger, Stefan Ropke, and Richard F. Hartl. The electric fleet size and mix vehicle routing problem with time windows and recharging stations. *European Journal of Operational Research*, 252:995–1018, 2016.

[276] M. Bruglieri, M. Paolucci, and O. Pisacane. A matheuristic for the electric vehicle routing problem with time windows and a realistic energy consumption model. *Computers & Operations Research*, 157, 2023.

[277] Michael Schneider, Andreas Stenger, and Dominik Goeke. The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science*, 48:500–520, 2014.

[278] Michail Masikos, Konstantinos Demestichas, Evgenia Adamopoulou, and Michael Theologou. Machine-learning methodology for energy efficient routing. *IET Intelligent Transport Systems*, 8:255–265, 2014.

[279] Cedric De Cauwer, Wouter Verbeke, Joeri Van Mierlo, and Thierry Coosemans. A model for range estimation and energy-efficient routing of electric vehicles in real-world conditions. *IEEE Transactions on Intelligent Transportation Systems*, 21:2787–2800, 2020.

[280] Xiaowen Bi, Andrew John Chipperfield, and Wallace K. S. Tang. Coordinating electric vehicle flow distribution and charger allocation by joint optimization. *IEEE Transactions on Industrial Informatics*, 17:8112–8121, 2021.

[281] Cristian Cataldo-Díaz, Rodrigo Linfati, and John Willmer Escobar. Mathematical model for the electric vehicle routing problem considering the state of charge of the batteries. *Sustainability*, 14, 2022.

[282] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[283] Gitae Kim, Yew-Soon Ong, Chen Kim Heng, Puay Siew Tan, and Nengsheng Allan Zhang. City vehicle routing problem (city vrp): A review. *IEEE Transactions on Intelligent Transportation Systems*, 16(4):1654–1666, 2015.

[284] Hoong Chuin Lau and Aldy Gunawan. The patrol scheduling problem. PATAT, 2012.

[285] Shangyao Yan, Chun-Yi Wang, and Yu-Wei Chuang. Optimal scheduling for police patrol duties. *Journal of the Chinese Institute of Engineers*, 43(1):1–12, 2020.

[286] Nicole Adler, Alfred Shalom Hakkert, Tal Raviv, and Mali Sher. The traffic police location and schedule assignment problem. *Journal of Multi-Criteria Decision Analysis*, 21(5-6):315–333, 2014.

[287] Paul A Chircop, Timothy J Surendonk, Menkes HL van den Briel, and Toby Walsh. On routing and scheduling a fleet of resource-constrained vessels to provide ongoing continuous patrol coverage. *Annals of Operations Research*, 312(2):723–760, 2022.

[288] Paul A Chircop and Timothy J Surendonk. On integer linear programming formulations of a patrol boat scheduling problem with complete coverage requirements. *The Journal of Defense Modeling and Simulation*, 18(4):429–439, 2021.

[289] Yue-Jiao Gong, Jun Zhang, Ou Liu, Rui-Zhang Huang, Henry Shu-Hung Chung, and Yu-Hui Shi. Optimizing the vehicle routing problem with time windows: A discrete particle swarm optimization approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(2):254–267, 2011.

[290] Jiahai Wang, Ying Zhou, Yong Wang, Jun Zhang, CL Philip Chen, and Zibin Zheng. Multiobjective vehicle routing problems with simultaneous delivery and pickup and time windows: Formulation, instances, and algorithms. *IEEE transactions on cybernetics*, 46(3):582–594, 2015.

[291] Gitae Kim, Yew Soon Ong, Taesu Cheong, and Puay Siew Tan. Solving the dynamic vehicle routing problem under traffic congestion. *IEEE Transactions on Intelligent Transportation Systems*, 17(8):2367–2380, 2016.

[292] Wanyuan Wang, Zichen Dong, Bo An, and Yichuan Jiang. Toward efficient city-scale patrol planning using decomposition and grafting. *IEEE Transactions on Intelligent Transportation Systems*, 22(2):747–757, 2019.

[293] Linhui Cheng, Luo Zhong, Shasha Tian, and Jiaxu Xing. Task assignment algorithm for road patrol by multiple uavs with multiple bases and rechargeable endurance. *IEEE Access*, 7:144381–144397, 2019.

[294] PA Chircop, TJ Surendonk, MHL van den Briel, and T Walsh. A column generation approach for the scheduling of patrol boats to provide complete patrol coverage. In *Proceedings of the 20th international congress on modelling and simulation*, pages 1–6, 2013.

[295] Leila Hajibabai and Debashis Saha. Patrol route planning for incident response vehicles under dispatching station scenarios. *Computer-Aided Civil and Infrastructure Engineering*, 34(1):58–70, 2019.

[296] Hoong Chuin Lau, Zhi Yuan, and Aldy Gunawan. Patrol scheduling in urban rail network. *Annals of Operations Research*, 239:317–342, 2016.

[297] Xingquan Zuo, Cheng Chen, Wei Tan, and MengChu Zhou. Vehicle scheduling of an urban bus line via an improved multiobjective genetic algorithm. *IEEE Transactions on intelligent transportation systems*, 16(2):1030–1041, 2014.

[298] Shinwon Choi, Minseo Lee, Hyejin Park, and Jinil Han. Mathematical programming-based heuristic for highway patrol drone scheduling problem. *Socio-Economic Planning Sciences*, 93:101907, 2024.

[299] Rafael Bautista Linde. Deep reinforcement learning for uav patrolling in rescue operations. 2023.

[300] Jimmy Perron, Jimmy Hogan, Bernard Moulin, Jean Berger, and Micheline Bélanger. A hybrid approach based on multi-agent geosimulation and reinforcement learning to solve a uav patrolling problem. In *2008 Winter Simulation Conference*, pages 1259–1267. IEEE, 2008.

[301] Waldy Joe and Hoong Chuin Lau. Learning to send reinforcements: coordinating multi-agent dynamic police patrol dispatching and rescheduling via reinforcement learning. 2023.

[302] Waldy Joe, Hoong Chuin Lau, and Jonathan Pan. Reinforcement learning approach to solve dynamic bi-objective police patrol dispatching and rescheduling problem. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, pages 453–461, 2022.

[303] Songhan Wong, Waldy Joe, and Hoong Chuin Lau. Dynamic police patrol scheduling with multi-agent reinforcement learning. In *International Conference on Learning and Intelligent Optimization*, pages 567–582. Springer, 2023.

[304] Karla L Hoffman, Manfred Padberg, Giovanni Rinaldi, et al. Traveling salesman problem. *Encyclopedia of operations research and management science*, 1:1573–1578, 2013.

[305] Juan Pablo Vielma. Mixed integer linear programming formulation techniques. *Siam Review*, 57(1):3–57, 2015.

[306] J Cole Smith and Z Caner Taskin. A tutorial guide to mixed-integer programming models and solution techniques. *Optimization in medicine and biology*, pages 521–548, 2008.

[307] Kim-Fung Man, Kit Sang Tang, and Sam Kwong. *Genetic algorithms: concepts and designs*. Springer Science & Business Media, 2001.

[308] Fred W Glover and Gary A Kochenberger. *Handbook of metaheuristics*, volume 57. Springer Science & Business Media, 2003.

[309] Aric Hagberg and Drew Conway. Networkx: Network analysis with python. *URL: https://networkx. github. io*, 2020.

[310] Geoff Boeing. Modeling and analyzing urban networks and amenities with osmnx. 2024.

[311] Majid Ghasemi and Dariush Ebrahimi. Introduction to reinforcement learning, 2024.

List of Symbols and Notations

This chapter provides a comprehensive list of the symbols and notations used throughout the thesis. The notations are categorized based on their respective chapters and applications.

Table 6.1: A list of notations and symbols used in Chapter 2

Symbols and Notations	**Definition**
S	A finite set of states
s	State
s'	Next state
A	A finite set of actions
a	Action
$P(s'\|s,a)$	Probability of transitioning to s' by taking action a
$R(s,a)$	The immediate reward received after taking action a in state s
γ	Discount factor
π	Policy
$V_\pi(s)$	State-value function
$Q_\pi(s,a)$	Action-value function
G_t	Expected cumulative reward
α	Learning rate
ϵ	Exploration rate
δ	Temporal difference (TD) error
$e(s)$	Eligibility trace
q_*	Optimal action-value function

Table 6.2: A list of notations and symbols used in Chapter 3

Symbols and Notations	**Definition**
θ	Neural network parameters
$L(\theta)$	Loss function for deep learning models
D	Experience replay buffer
y_t	Target for updating neural network
ϵ	Exploration rate in ϵ-greedy policy
$J(\pi)$	The expected discounted reward for a policy π
$D_{KL}(\pi_\theta \| \pi_{\theta'})$	Kullback-Leibler (KL) divergence between the new policy $\pi_{\theta'}$ and the old policy π_θ
$A(s, a)$	Advantage function
$Q^\mu(s, a)$	Action-value function under μ_θ
Q_θ	Critic network
$Q_{\theta'_i}$	Target Critic network
μ_θ	Actor network
$\mu_{\theta'}$	Target actor network

Table 6.3: A complete list of notations and symbols used in Chapter 4

Symbols and Notations	**Definition**
$G = (N, E)$	Graph representing the transportation network with nodes N and edges E.
N	Set of nodes, including intersections and charging stations.
E	Set of edges representing roads connecting nodes.
S	Source node.
D	Destination node.
$I_{station}$	Set of charging stations.
$T_{edge}(i, j)$	Travel time on edge (i, j).
B_s	Initial battery level of the electric vehicle (EV).
B_i	Battery level of the EV at node i.
B_{min}	Minimum battery level required to continue the journey.
B_{max}	Maximum battery capacity.
$E_{depletion}$	Energy depletion rate per unit distance.
$E_{charging}$	Energy recharge rate at charging stations.
T_{charge}	Time required to charge at a station.
P_{energy}	Energy consumption per unit distance.
S_{charge}	Available charging slots at a station.
$P_{traffic}$	Traffic congestion factor affecting travel time.
$Q(s, a)$	Q-value function for action a in state s.
$R(s, a)$	Reward function for taking action a in state s.
$\pi(s)$	Policy mapping states to actions.
γ	Discount factor for future rewards.
α	Learning rate in reinforcement learning.
ϵ	Exploration rate in the ϵ-greedy policy.
TD	Total travel duration including charging stops.
$L(\theta)$	Loss function in deep reinforcement learning.
θ	Neural network parameters.
y_t	Target value for updating neural networks.
D	Experience replay buffer.
τ	Target network update rate.
$J(\pi)$	Expected cumulative reward for policy π.

Table 6.4: A complete list of notations and symbols used in Chapter 5

Symbols and Notations	**Definition**
$G = (N, E)$	Graph representing the patrol area, where N is the set of nodes and E is the set of edges.
N	Set of patrol locations, including the depot.
E	Set of edges representing travel paths between locations.
C	Set of available patrol vehicles.
D	Depot location index.
S	Set of patrol shifts.
T_{shift}	Maximum operation time per shift.
T_{patrol}	Minimum patrolling time required at each location.
T_{gap}	Maximum allowable time between consecutive visits to the same location.
T_{rest}	Minimum required rest time for a vehicle.
$W_{l,l'}$	Travel time between locations l and l'.
V_l	Total number of visits required for location l.
$X_{c,l,v} \in \{0, 1\}$	Binary decision variable indicating whether vehicle c has completed visit v at location l.
$Y_{c,l,v,s} \in \{0, 1\}$	Binary variable indicating whether vehicle c has completed visit v at location l during shift s.
$A_{c,v}$	Start time of the v-th visit by vehicle c.
$P_{c,v}$	Patrolling time for vehicle c on its v-th visit.
$L_{l,s}$	Time at which location l was last visited in shift s.
$Z_{gap_{c,l,v,s}}$	Decision variable used for enforcing the consecutive visits gap constraint.
$Z_{patrol_{c,l,v,s}}$	Decision variable ensuring that a minimum patrolling time is maintained.
$P_{emergency}$	Probability of an emergency occurring at a given location.
SST	Start time of a shift.
SET	End time of a shift.
CT	Current time during the patrol schedule execution.
CS	Current solution in the optimization process.
CSO	Objective value of the current solution.
$CSObetter$	Improved objective value found during optimization.
$Croute$	The currently selected patrol route for optimization.

List of Abbreviations

This chapter provides a comprehensive list of abbreviations used throughout the thesis. The abbreviations are arranged alphabetically for ease of reference.

Table 6.5: List of Abbreviations Used in This Thesis (Part 1)

Abbreviation	**Definition**
A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
AHBPS	Adaptive Hill-Climbing-Based Patrol Scheduling
AHBPS2	(Improved) Adaptive Hill-Climbing-Based Patrol Scheduling
CNN	Convolutional Neural Network
CFCU	Central Fleet Control Unit
DDPG	Deep Deterministic Policy Gradient
DDQN	Double Deep Q-Network
DL	Deep Learning
DP	Dynamic Programming
DQL	Deep Q-Learning
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
EV	Electric Vehicle
GDVPS	Genetic-Based Dynamic Vehicle Patrol Scheduling
GA	Genetic Algorithm
KL	Kullback-Leibler Divergence
LSTM	Long Short-Term Memory
MC	Monte Carlo
MDP	Markov Decision Process
MILP	Mixed Integer Linear Programming
MLP	Multi-Layer Perceptron

Table 6.6: List of Abbreviations Used in This Thesis (Part 2)

Abbreviation	**Definition**
MOMA	Multi-Objective Metaheuristic Algorithm
MOLS	Multi-Objective Local Search
NLP	Natural Language Processing
OSM	Open Street Maps
OP	Optimization Problem
$\mathcal{P}^4\mathcal{O}$	Patrol Planning with Proximal Policy Optimization
PPO	Proximal Policy Optimization
PSO	Particle Swarm Optimization
RL	Reinforcement Learning
SARSA	State-Action-Reward-State-Action
TD	Temporal Difference
TD3	Twin Delayed Deep Deterministic Policy Gradient
Dynamic TERC2	Dynamic Time Efficient Routing & Charging
TQL	Terc2 Q-Learning
TRPO	Trust Region Policy Optimization
TSP	Traveling Salesman Problem
UAV	Unmanned Aerial Vehicle
VPS	Vehicle Patrol Scheduling