# DEEP REINFORCEMENT LEARNING APPROACH TO MINIMIZE TASK WAIT TIMES IN DYNAMIC VEHICLE ROUTING

**A THESIS**

*submitted in partial fulfillment of the requirements*

*for the award of the dual degree of*

**Bachelor of Science - Master of Science**

*in*

**DATA SCIENCE AND ENGINEERING**

*by*

**SASWATA SARKAR**

**(19279)**

**DEPARTMENT OF DATA SCIENCE AND ENGINEERING**
**INDIAN INSTITUTE OF SCIENCE EDUCATION AND RESEARCH BHOPAL**
**BHOPAL - 462066**

**April 2024**

भारतीय विज्ञान शिक्षा एवं अनुसंधान संस्थान, भोपाल
Indian Institute of Science Education and Research, Bhopal
Department of Data Science and Engineering
Bhopal Bypass Road, Bhauri Bhopal 462 066

# CERTIFICATE

This is to certify that **Saswata Sarkar**, BS-MS (Dual Degree) student in the Department of Data Science and Engineering, has completed bonafide work on the thesis entitled **'Deep Reinforcement Learning Approach to Minimize Task Wait Times in Dynamic Vehicle Routing'** under my supervision and guidance. The content of this report is original and has not been submitted elsewhere for the award of any academic or professional degree.

**April 2024**
**IISER Bhopal**

**Prof. P. B. Sujit**

**Dr. Akshay Agarwal**

| Committee Member | Signature | Date |
|---|---|---|
| Dr. Vaibhav Kumar | | |
| Dr. Jasabanta Patro | | |
| Dr. Vinod K Kurmi | | |

# ACADEMIC INTEGRITY AND COPYRIGHT DISCLAIMER

I hereby declare that this project report is my work, and due acknowledgment has been made wherever the work described is based on the findings of other investigators. This report has not been accepted for the award of any other degree or diploma at IISER Bhopal or any other educational institution. I also declare that I have adhered to all academic honesty and integrity principles and have not misrepresented, fabricated, or falsified any idea/data/fact/source in my submission.

I certify that all copyrighted material incorporated into this document complies with the Indian Copyright (Amendment) Act (2012) and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and safeguard IISER Bhopal from any claims that may arise from any copyright violation.

**April 2024** **Saswata Sarkar**
**IISER Bhopal**

# ACKNOWLEDGMENT

To begin with, I express my sincere gratitude to Prof. P. B. Sujit, whose invaluable role in my undergraduate journey cannot be put into words. His mentorship has been a consistent and invaluable source of support, offering me guidance and resources whenever needed throughout my undergraduate journey. Since our encounter in my junior year, his kindness and motivation have inspired me to strive for excellence in research and personal blossoming. I will forever be indebted to him for his staunch faith in my potential.

In addition, I extend my heartfelt appreciation to Dr. Akshay Agarwal for showing his support and trust during my final year. His work ethic and morals have always been an inspiration to me. I am fortunate to have been an MS student under his guidance.

I also feel blessed to have learned from distinguished scientists such as Dr. Nabamita Banerjee, Dr. Anjan Gupta, Prof. Sebastian Wüster, and Prof. Deepak Chopra. Inside and outside the classroom, they have always addressed my doubts with great enthusiasm and care, supplementing my undergraduate journey and nurturing my growth as a science student.

It warms my heart to thank Prarthana for her tireless care and support through the ups and downs. I thank Saikat, Sharanyak, Mandi, Rwik, Nihal, Mizan, and Maharnab for their continuous companionship in helping me overcome the challenges of academic pursuits. I am equally thankful to my peers at MOON Lab, including Manav, Swadhin, Karthik, Vinita, and Rajat, as well as my seniors - Anirban, Pramanik, Debjit, Kasi, Jit, Rivu, Prithvi, Rana, and Sayan.

I am deeply grateful to all the teachers and classmates from Amar Krishna Pathsala, who helped nurturing my young mind and inspiring me to pursue a career in science.

Lastly, I express my deepest gratitude to my parents, Mrs. Dola Sarkar and Mr. Susanta Sarkar, and my beloved sister, Swastika. Their endless love and relentless trust have been my greatest supporters throughout my five-year journey at this esteemed institution. I dedicate this work to my Bomma and seek her blessings for my next endeavors.

**Saswata Sarkar**

# ABSTRACT

Vehicle Routing is a critical operational challenge faced in various sectors, such as logistics and transportation, requiring efficient real-time route optimization to manage dynamically arriving task demands. While contemporary works often explore methods to optimize the total distance traveled, they often overlook the punctual completion of incoming tasks in time-sensitive domains, which is equally important. This thesis explores the application of Deep Reinforcement Learning to build an alternate policy to minimize task wait times in DVR, focusing on a single-agent scenario. The research employs a novel framework that integrates the Proximal Policy Optimization algorithm within a custom simulated environment to address the stochastic nature of task arrivals. Extensive simulations demonstrate that the proposed Deep-RL approach significantly outperforms baseline greedy heuristics, yielding at least 46% more efficient solutions on average in optimizing task wait times by efficiently managing route planning and task prioritization. This study shows how Deep-RL can improve real-world vehicle routing tasks, highlighting the benefits of using advanced AI techniques to enhance operational efficiencies in complex environments. Our code is available at https://github.com/sarkarsaswata/Dynamic-VRP.

**Keywords:**

**Deep-RL, Single-agent, Route optimization, Task wait time**

# LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| VR | Vehicle Routing Problem |
| SVR | Static Vehicle Routing |
| DVR | Dynamic Vehicle Routing |
| ML | Machine Learning |
| Deep-RL | Deep Reinforcement Learning |
| $\rho$ | Load factor |
| $\lambda$ | Task arrival rate |
| $\bar{t}$ | Mean service time of tasks |
| TSP | Traveling Salesman Problem |
| $\mathcal{S}$ | Set of states |
| $\mathcal{A}$ | Set of actions |
| $\mathcal{T}$ | State transition function |
| $\mathcal{R}$ | Set of rewards |
| $\pi$ | Policy (decision making rule) |
| $\gamma$ | Discount factor |

# LIST OF FIGURES

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

Vehicle routing (VR) is a crucial aspect of many real-life applications, such as parcel delivery services [1], transportation of goods [2] in warehouse, and waste collection [3]. For instance, companies like Amazon and FedEx rely on efficient vehicle routing that optimizes total traveled distance to deliver packages to customers promptly. Similarly, waste collection services utilize vehicle routing to optimize routes and minimize fuel consumption while collecting all waste on time. In Static Vehicle Routing (SVR), a fleet of vehicles from a central depot must service a set of customers with known demands and locations. The goal is to optimize the routes to reduce the total distance traveled by the vehicles while ensuring that each customer's demand is satisfied. SVR is a deterministic problem, meaning all the task-related information is known beforehand.

Dynamic Vehicle Routing (DVR) belongs to a subset of combinatorial optimization challenges wherein the goal is to devise optimal routes and schedules for a caravan of autonomous vehicles assigned to fulfill a range of tasks, with task requirements changing over time [4]. DVR presents additional challenges compared to SVR, as it requires real-time decision-making and adaptation to the dynamics of the environment. Real-world applications of DVR have recently garnered considerable attention in diverse fields, including surveillance, environmental monitoring, and disaster management [5], as the use of UAVs and drones becomes emerging. Traditional approaches to solving variants of DVR employ dynamic programming, evolutionary algorithms, meta-heuristics, and

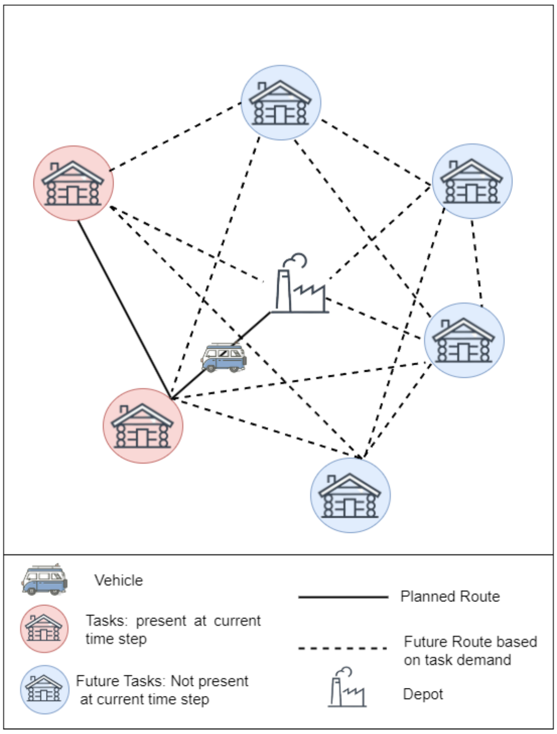

Vehicle
Tasks: present at current time step
Future Tasks: Not present at current time step
Planned Route
Future Route based on task demand
Depot

Figure 1.1: **Overview of Single Vehicle Dynamic Vehicle Routing**: Unlike Static VR, the key consideration is a stochastic task arrival process. A single vehicle is assigned to service all incoming tasks. In the illustration, red houses indicate current tasks, and the straight lines depict the planned route. Blue houses represent stochastic task arrivals, necessitating new routes (shown as dashed lines.)

integer linear programming, providing theoretical guarantees [6, 7, 8, 9, 10]. Existing algorithms recompute routes based on service availability and task demand, which is repetitive and computationally exhausting and thus lacks repetitive time adaptability [11, 12, 13]. If artificial intelligence (AI) can implement automation in dynamic decision-making, we can counter the dynamics inherent to DVR. In this thesis, we advocate using a deep reinforcement learning (Deep-RL) approach to address this challenge and improve the efficiency of DVR.

RL is a subtype of machine learning (ML), well-suited for sequential decision-making [14] in dynamic and uncertain environments [15]. Further progress on leveraging Deep Neural Networks (DNNs) in RL has shown superhuman-level control on Atari [16], complex board games like playing Go [17], and real-time-strategy (RTS) benchmarks like `DOTA 2` [18], `FLATLAND` [19], and `STARCRAFT II` [20]. There have been advancements in utilizing Deep Reinforcement Learning (Deep-RL) and dynamic attention models to tackle DVR more effectively [21]. Henceforth, utilizing Deep-RL techniques to solve DVR with offline training[1] to optimize task wait times and improve overall performance while handling stochastic task arrival requests seems promising. This thesis addresses the challenges of optimizing the wait times of tasks in the DVR setup by leveraging the Deep Reinforcement Learning (Deep-RL) framework.

## 1.2 Dynamic Vehicle Routing

Dynamic Vehicle Routing, a long-studied NP-Hard problem [22] in operations research, has significant implications for the efficiency and cost-effectiveness of transportation and logistics systems [2]. Applications of DVR encompass, though are not limited to, parcel delivery [1], ride-sharing [23], waste collection [3], and forest fire monitoring [24]. The typical scenario in DVR involves route planning and task scheduling for one or more autonomous vehicles in real-time, where the planning needs to adapt on the fly based on stochastic task arrival patterns and changing dynamics of the environment. In Fig. 1.1, we illustrate the idea of dynamic vehicle routing where the key objective is to minimize costs associated with vehicle routing, such as total distance traveled or total time taken [25]. While previous studies have primarily focused on reducing TSP tours[2], it is imperative to address the timely completion of tasks in time-sensitive systems. This study will emphasize a specific problem within the DVR domain, which involves managing a single vehicle's *route planning* and *task scheduling* in real time, considering a stochastic task arrival process and minimizing task wait times.

When examining the DVR with stochastic task arrival process, it is essential to consider the load factor $\rho$, defined in the range $(0, 1)$, as pointed out in [6] and [26]. The load factor $\rho$ for DVR derives from Equation 1.1. The Equation 1.1 establishes a relation between the task arrival rate ($\lambda$), average service time for the tasks ($\bar{t}$) and the number of vehicles ($n$). The load factor indicates how much time vehicles must spend to attend to incoming tasks.

[1] Offline training, in RL, refers to training agents using simulation data.

[2] TSP tours, a Hamiltonian cycle referring to the shortest tour where all the task nodes are visited at least once

$$\rho = \frac{\lambda \cdot \bar{t}}{n} \tag{1.1}$$

In low load, characterized by $\rho \to 0^+$, the length of the unserved task queue tends to zero, allowing vehicles ample idle time to wait for incoming tasks. Alternatively, vehicles must continuously service tasks in high-load situations where $\rho \to 1^-$, resulting in an indefinitely long queue for unserved tasks [6]. Both scenarios present operational challenges: in low-load conditions, vehicles operate at low capacity, while in high-load situations, task wait times become unacceptably long. The system opera

Previous studies on the DVR to develop exact and heuristic algorithms have mainly focused on guarantees in extreme load conditions [1, 11, 23, 27, 28], often neglecting the space where the system operates optimally. Notably, the load factor $(\rho)$ should be less than $0.9$ [29, 30], due to the operational advantage, as the length of the pending tasks $(L)$ grows in accordance with $\frac{1}{(1-\rho)^2}$. This thesis focuses on the medium-load DVR for a single vehicle, where $\rho \in [0.5, 0.9]$, as the unserved task queue, neither approaches zero nor infinity. We visualize the relation of load factor and unserved task queue in Fig. 1.2 assuming the proportional constant to be $k = 1$.

This thesis presents a Deep-RL approach to tackle the intricate and stochastic setting of DVR under medium-load conditions, with a specific emphasis on both path planning and control to optimize task wait times. The proposed solution advances the utilization of the Deep-RL algorithm to solve the challenges posed by DVR. The critical focus of the proposed solution is to minimize the average wait times for tasks to enhance service quality and operational efficiency.

**Definition 1. Task Wait Time**

Task wait time for task $i$, represented as $W_i$, refers to the duration between a task $i$ entering the system and being served by a vehicle.

**Definition 2. Load Factor ($\rho$)**

The load factor, falling within the range of $(0, 1)$, is a measure computed by dividing the product of the task arrival rate $(\lambda)$ and the mean task servicing time $(\bar{t})$ by the number of vehicles $(n)$. [Equation (1.1)]

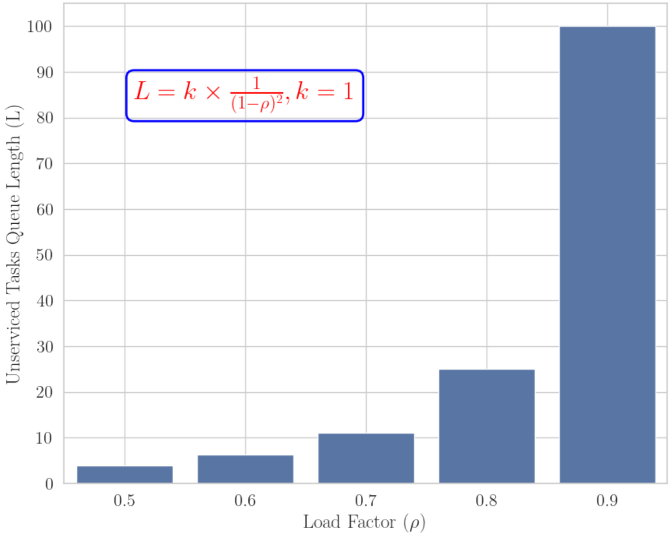


Figure 1.2: Relation between load factor and unserved task queue length, assuming the proportional constant as $k = 1$. As the load factor increases, the queue of unserved tasks becomes indefinitely long compared to a lower load factor.

## 1.3 Key Contribution

The primary contributions of this thesis encompass:

- Formulating the Markov Decision Process (MDP) for Dynamic Vehicle Routing (DVR) with stochastic task arrival process as a Deep Reinforcement Learning (Deep-RL) problem.
- Adoption of minimal observation instead of route-based MDPs to address DVR.
- Developing a novel reward function to optimize the mean wait time of stochastic tasks.

In the subsequent sections, we survey the contemporary research concerning a range of diverse variants of DVR. Through meticulous analysis, we uncover the strengths and limitations of these studies, offering valuable insights into the evolving landscape of DVR. Building upon this foundation, we delve into the intricacies of modeling DVR as a Markov Decision Process (MDP), highlighting the dynamic nature of the stochastic task arrival process. This theoretical framework serves as the basis of our methodology, providing a structured approach to our research. Subsequently, we elucidate the methods we employ, with a particular focus on our innovative utilization of Deep Reinforcement Learning (Deep-RL).

# Chapter 2

# Literature Review

This section seeks to set the context, pinpoint knowledge gaps, and underscore the importance of this study by offering a thorough analysis of prior research in the field of DVR. Through a meticulous assessment of previous works, including studies, theories, and methodologies, this section lays the foundation for the thesis work, guiding its direction and contributing to advancing knowledge in the field.

## 2.1 Related Works

In DVR, incoming tasks follow a stochastic arrival pattern where a Poisson process governs the inter-arrival time of the tasks, and the task locations are distributed uniformly over the environment. DVR expands the challenge of Traveling Salesman Problem (TSP) to incorporate the dynamic intricacies of real-world vehicle routing into the picture [22]. Over the past decades, researchers have rigorously studied numerous variants of DVR [31], demonstrating the significance and complexity of the optimization challenges involved. Bertsimas and Ryzin first introduced the consideration of low and high-load DVR [6]. Researchers have proposed different mathematical models and algorithms ranging from exact algorithms to heuristic and meta-heuristic algorithms to address the complexities of DVR [4]. For instance, Bertsimas and Ryzin in [30] introduced the Dynamic Traveling Repairman Problem (DTRP) as a generic model for dynamic and stochastic VR problems.

Real-time information, such as task demands, is crucial in solving DVR efficiently. Strategies like selecting alternate routes, switching the order of task visits, and adapting allocation vehicle-task pair based on real-time data have emerged to optimize vehicle routing under dynamic conditions [7]. Exact algorithms use integer linear programming formulation and techniques like branch and cut or dynamic programming to find optimal solutions. On the flip side, heuristic methods, including classical and meta-heuristics, have gained prominence for their effectiveness in solving Vehicle Routing instances. Classical heuristics [8] focus on incremental improvements within local neighborhoods, gradually refining solutions until further gains are no longer possible. Conversely, meta-heuristic approaches offer a more flexible strategy, disabling exploration of the non-improving or infeasible solutions to overcome computational challenges associated with DVR [9, 10]. Pavone et al. proposed a new `BATCH` policy algorithm [13], which computes TSP tours for a batch of incoming new tasks and randomly selects a portion of the batch for service during each iteration.

Researchers have explored diverse strategies and advancements to reduce task wait times in DVR. One approach combines the use of deterministic wait strategies along with evolutionary algorithms to optimize wait times empirically [5]. Additionally, researchers have introduced the concept of degree dynamism to address the stochasticity inherent to the routing problem, aiming to minimize the expected wait time of tasks [12]. Recent progress in the domain has concentrated on leveraging frequent data and cutting-edge technologies to augment routing effectiveness. The amalgamation of Intelligent Transport Systems (ITS), Fleet Management Systems (FMS), and Global Positioning Systems (GPS) has facilitated the evolution of DVR, sparking a notable increase in associated research endeavors [5].

Moreover, researchers have proposed adaptive-distributed algorithms in changing environments [32]. Further, Botros et al. improved the `BATCH` policy [13] by introducing a novel cost function to optimize average and maximum task wait times [33]. Furthermore, a recent study has emphasized the importance of anticipation and proactive re-optimization in DVR with stochastic requests to improve routing efficiency [34]. Researchers have highlighted the dynamically adjusted travel time as a critical element in saving energy and reducing wait times in route planning for electric vehicles [35]. Additionally, fuzzy receding horizon control strategies have shown promise in addressing DVR challenges [36].

The latest developments in neural computing have also brought attention to the utilization of Deep RL techniques (Deep-RL) in Dynamic Vehicle Routing (DVR), offering automation for both path planning and scheduling tasks [21, 37, 38, 39]. Recent research has introduced an online approach that combines RL with a non-myopic solver, which considers future requests to make more informed decisions [28].

## 2.2 Research Gaps

The preceding review provides a comprehensive overview of research efforts in Dynamic Vehicle Routing (DVR), highlighting established methodologies and emerging trends. Previous studies have extensively explored various aspects of DVR, including considering extreme load scenarios, developing mathematical models, and implementing algorithms ranging from exact solutions to heuristic and meta-heuristic approaches. However, despite the progress in optimizing routing efficacy and minimizing TSP tours, there are notable research gaps in exploring alternate policies to reduce task wait times in time-sensitive systems that need further investigation.

One significant research gap lies in applying Deep-RL to address DVR challenges, particularly in reducing task wait times for medium loads scenarios. While existing RL solutions have utilized complex Route-Based Markov Decision Process (RBMDP) formulations as described in the work [40], the MDP formulation can be streamlined by focusing on minimal information for state representation, such as task locations and the corresponding wait times. Moreover, integrating real-time information and advanced technologies, such as ITS and GPS, presents opportunities for enhancing vehicle routing efficiency.

In light of these research gaps, this thesis aims to explore alternative Deep-RL policies explicitly tailored to DVR, primarily focusing on minimizing task wait times and enhancing service quality. By delving into unexplored avenues within the intersection of Deep-RL and DVR optimization, this study seeks to contribute insights and methods to the field, ultimately advancing the SOTA[1] in dynamic vehicle routing and addressing the evolving challenges of modern transportation systems.

[1] state-of-the-art

# Chapter 3

# Modeling DVR

This section delves into the foundational principles of a Markov Decision Process (MDP), a mathematical framework for modeling agent-environment interaction in reinforcement learning (RL). Subsequently, we establish the problem statement and outline the fundamental assumptions that underpin the thesis.

## 3.1 Markov Decision Process and RL

A Markov Decision Process (MDP) [41] is a mathematical framework to model an agent interacting with the environment. In Fig. 3.1, we visualize the agent-environment interaction as an MDP. It models a decision-making RL problem in the form of a tuple $\mathcal{G} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, and $\mathcal{R}$ is the set of rewards. We define the dynamics of the MDP with a stochastic state transition function, $\mathcal{T} : p(s', r|s, a) \doteq \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$, that maps a state and an action of the decision-making agent to a probability distribution over the next states. We define the reward function $R : S \times A \to \mathbb{R}, \forall S \in \mathcal{S}, \forall A \in \mathcal{A}$, that maps a state-action pair to a scalar value $r \in \mathbb{R}$. The observation for the agent finds representation by using the transition function, encompassing all the information the agent obtains from taking an action in the environment.

The agent uses the state at the current step $s_t$ to select an action according to its stochastic policy $\pi(s_t)$. The policy of a Deep-RL agent takes the state $s_t$ of the agent as input and outputs the probability distribution over the possible actions of the agent for

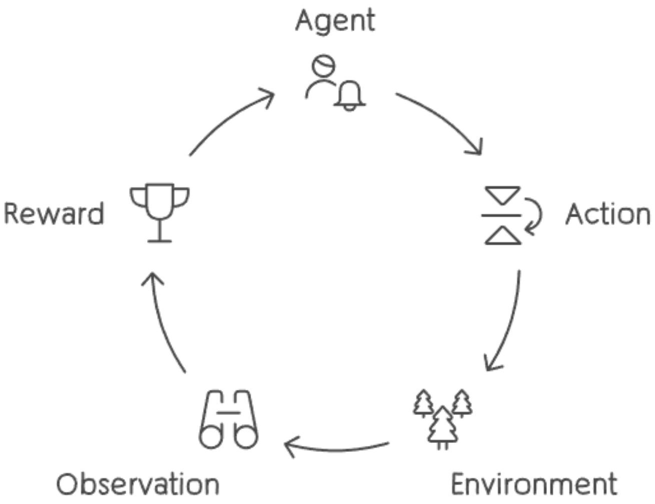


Figure 3.1: Interaction between Agent and Environment in a Markov Decision Process. Agent takes an *action* in the environment and receives *observation* and *reward* as a feedback.

the next step. We represent the policy of our Deep-RL agent as $\pi : S_t \rightarrow \Delta(A_t)$, where $\Delta(A_t) = \{\pi(a_t|s_t) : a_t \in A_t\}$ is the set of probability distributions over the action space $A_t$. The return[1] in state $s_t$ with discount factor $\gamma \in [0, 1)$ is as follows, where $s_0$ is the terminal state:

$$R(s_t) = \sum_{t=0}^{T} \left[\gamma^t r_t \left(s_0, a_1, \cdots, a_T \mid s_0 = s_t\right)\right] \tag{3.1}$$

Considering a finite horizon episode of length $T$, then, the objective function for the Deep-RL agent to maximize represents as the following:

$$J = \mathbb{E}_\pi \left[\sum_{t=1}^{T} \gamma^t r_t\right] \tag{3.2}$$

Here, $\mathbb{E}$ represents the expected value of the cumulative discounted reward for the discount factor $\gamma \in (0, 1)$ following the policy $\pi$. In Section 4.2, a detailed explanation of the state and action space formulated for the DVR at hand provides a comprehensive understanding of these fundamental components.

[1] **Return** is the cumulative sum of rewards the agent expects to gain when following the current policy starting at the current state to the terminal state.

## 3.2 Problem Formulation

This section revisits the formulation of DVR within a bounded and convex spatial domain $\mathcal{E} \subset \mathbb{R}^2$, representing a two-dimensional Euclidean plane $d = 2$. In this context, a vehicle, operating at a consistent unit velocity $v$, attends to stochastically occurring task arrivals within $\mathcal{E}$. The task arrivals conform to a Poisson process with a time-density parameter $\lambda \in \mathbb{R}^+$. Notably, the tasks' arrivals are i.i.d.[2] in the environment $\mathcal{E}$, and servicing each task necessitates the vehicle to traverse to the task location. We denote the time needed to serve a task $i$ at its designated location as $t_i$. The system's load factor, as pictured in Equation 1.1, encapsulates the relationship between task arrival rate, service time, and the number of vehicles. For the DVR setup at hand, $\rho$, reduces to Equation 3.3, as we assume the average task service time is $\bar{t} = 1$, and a single vehicle $(m = 1)$ in the environment. Therefore, we can use the task arrival rate $\lambda$ of the stochastic process to model the load factor $\rho$ of the DVR.

$$\rho = \lambda \tag{3.3}$$

In the context of DVR, the wait time for a task $i$, as described in the definition 1, is $W_i$. If there are $n$ tasks to service in a finite time limit $T$, one calculates the average wait time for the entire system as $\overline{W} = \frac{1}{n}\sum_{i=1}^{n} W_i$. We define a tour $\tau$ as the sequence of visiting task locations as new tasks appear stochastically in the environment from time $0 \rightarrow T$. We define the optimal tour, denoted as $\tau'$, as the one that minimizes the mean wait time of all tasks to $\overline{W}'$. The objective of the thesis is to develop a Deep-RL policy that controls the vehicle's planning and navigation within a finite horizon episode of length $T$ to service all tasks while minimizing the system's average wait time, denoted as $\overline{W}$, such that $\overline{W} \rightarrow \overline{W}'$.

**Problem:**

Given a closed and convex two-dimensional environment $\mathcal{E} \subset \mathbb{R}^2$, a single vehicle, and tasks arriving governed by a Poisson process with mean $\lambda$, the objective is to determine an optimal RL policy, denoted as $\pi'$, which computes a tour $\tau'$minimizing the average wait time of the system to $\overline{W}'$.

[2] Independent and Identically Distributed

## 3.3 Key Assumptions

These presuppositions form the foundational principles upon which subsequent analyses and methodologies rely, guiding the solution of the DVR.

- Tasks arrive in the environment following a Poisson process.
- The average task service time is $\bar{t} = 1$.
- There is only one vehicle with enough fuel to service all the tasks within a finite time limit of $T$.

In subsequent sections, we will discuss the methodologies and results found in this research. We will also use ”vehicle” and ”agent” or ’Deep-RL agent” interchangeably to denote the entity responsible for decision-making and interaction within the context of dynamic vehicle routing problems.

# Chapter 4

# Methodology

In this thesis, we used the Proximal Policy Optimization algorithm [42] to learn a Deep-RL policy that navigates the RL agent (vehicle) and plans the task completion schedules for the DVR. Proximal Policy Optimization (PPO) is a SOTA, on-policy Deep-RL algorithm that balances performance and sample efficiency, making it fit for a broad range of problems compared to other RL algorithms. Studies have demonstrated that PPO [42] outperforms other widely used Deep-RL algorithms like DDPG [43], SAC [44], TD3 [45], particularly in complex control tasks. Here are the two main reasons for selecting the PPO algorithm to solve DVR.

- **Robustness:** PPO [42] shines in balancing sample efficiency and computational complexity, outperforming Policy-Gradient [46], Actor-Critic [47], and TRPO [48] methods, which have a reputation for high variance and expensive second-order optimization.

- **Stability:** PPO [42] is more stable and reliable than previous algorithms like TRPO [48], with improved data efficiency and better performance in continuous problems.

Furthermore, the work developed a novel state and action space suitable for the agent to learn optimal policies for dynamic routing. In the following sections, we will elaborate more on the selection of the state-action space, the reward signal that helps the agent understand the training goal, and how tweaking the reward signal changes the agent's behavior.

## 4.1 Proximal Policy Algorithm

PPO [42] is an on-policy Deep-RL algorithm that addresses the exploration-exploitation trade-off by optimizing a clipped surrogate objective function as described by the Equation 4.1. PPO's design aims to achieve greater data efficiency and robustness compared to earlier policy gradient methods while upholding the efficient interaction and reliable performance characteristic of TRPO [49]. The fundamental concept driving PPO is optimizing a clipped surrogate objective function, serving as a lower bound on the policy's performance. PPO seeks to balance exploration and exploitation by optimizing this clipped surrogate objective function. This function quantifies the disparity between the new and old policies, incorporating a clipping term to restrict the size of policy update steps. Consequently, PPO circumvents the potential pitfalls of large policy updates due to a stochastic environment, often leading to instability and subpar performance. Instead, PPO opts for smaller, more controlled policy updates, thus ensuring the maintenance of the current policy's performance while facilitating exploration and enhancement.

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[ \min \left( r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right] \tag{4.1}$$

Here, $r_t(\theta)$ is the probability ratio of the new policy $\pi$ to the old policy $\pi_{\text{old}}$, expressed as $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$. $\hat{A}_t$ denotes the advantage function, and $\theta$ represents, the set of parameters of the policy. PPO involves sampling data from the policy and performing multiple epochs of optimization on the sampled data. The algorithm alternates between sampling and optimization, allowing multiple updates per sampled data.

## 4.2 States and Action Space

Following the notations established in Section 3.1, the state space $\mathcal{S}$ and action space $\mathcal{A}$ are defined in the following subsections.

### 4.2.1 State Space

The state space, denoted as $\mathcal{S} \triangleq \{\ell_{\mathbf{a}}, \ell_{\mathbf{t}}, \mathbf{c_t}, \mathbf{p}\}$, encapsulates the informations from environment at any given time step $t$, represented as $s_t$. Assuming a set of $n$ tasks to be serviced, designated as a finite set $\{\ell_{t_1}, \ell_{t_2}, \cdots, \ell_{t_n}\}$, the constituents of $\mathcal{S}$ define as follows:

- $\boldsymbol{\ell_a}$ - An array of shape (2,) indicating the (x, y) coordinates of the agent's position at any timestep $t$.

- $\boldsymbol{\ell_t}$ - An array of shape (2, n) where the $j^{th}$ index represents the (x, y) coordinates of task $j$ present in the environment at any time step $t$. A value of 0 in the array signifies the absence of a task at that index at the timestep $t$.

- $\mathbf{c_t}$ - An array tracking the clock readings of tasks present at the current time step, where each value $c_{t_j} \in \mathbb{R}_{\geq 0}$ corresponds to the clock reading of task $j$.

- $\mathbf{p}$ - A scalar value within the range $[-1, 1]$ to monitor an episode's total elapsed time steps. Specifically, $p = -1$ corresponds to $t = 0$, and $p = 1$ represents $t = T$, where $T$ denotes a non-zero real number signifying the maximum time step in an episode.

As elucidated by Pardo et al. [50], we introduce the concept of a progress parameter within the state space ($\mathcal{S}$). This parameter serves the dual purpose of resetting the environment for subsequent training when the maximum time limit $T$ is reached and facilitating bootstrapping from the truncated state at the end of a partial episode (the one when we do not get the terminal state). We employ bootstrapping to update the value function upon reaching the terminal state while refraining from bootstrapping otherwise.

### 4.2.2 Action Space

A custom grid world environment as depicted in the Fig. 4.1 is developed to address the problem at hand, using the OpenAI Gym library [51]. Within this gym environment, the agent can execute discrete actions at each time step from the action space $\mathcal{A} = \{1, 2, \cdots, 8\}$. Each discrete action enables the agent to move its position to one of its adjacent cells, as illustrated in Fig. 4.2.

In this framework, we refrain from employing continuous state and action spaces to mitigate the exponential expansion of the search space for the Deep-RL agent. Instead, we assume the presence of a low-level planner within the system responsible for navigating the agent from one cell to another.

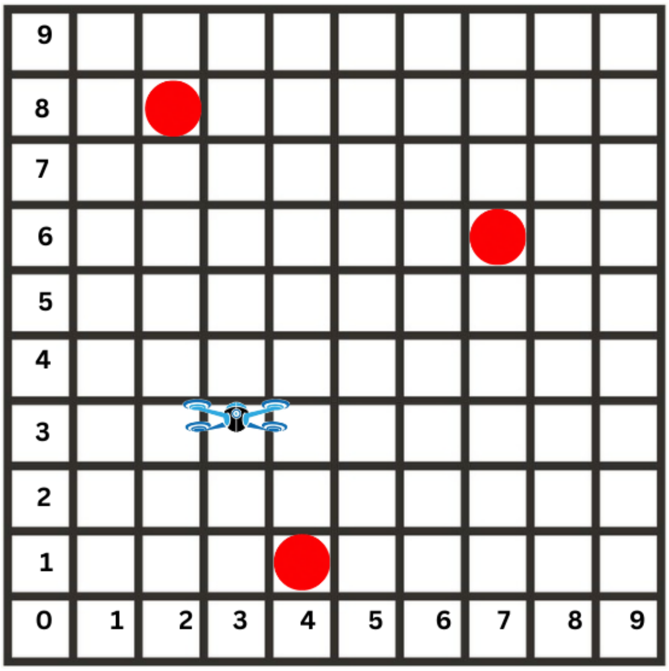


Figure 4.1: A bird's-eye view of the OpenAI Gym environment depicting the Grid-World. The drone indicates the agent's position within the grid cells. The red circles at cells within the environment denote the presence of unserved tasks.

### 4.2.3 Reward Function

In the experimental framework, a novel reward structure incentivizes the agent towards efficient target visitation while minimizing task wait times. Formally, we define the reward function as:

$$r(t) = \begin{cases} 2 \cdot e^{\left(1-\frac{\text{wait\_time}_j}{\text{max\_wait\_time}}\right)} & \text{if the agent services a task } j \text{ at timestep } t \\ -0.005 & \text{otherwise} \end{cases} \tag{4.2}$$

In this reward formulation, a timestep penalty of $-0.005$ is integrated to discourage aimless exploration by the agent and to promote swift completion of the episode. Additionally, the formulation incentivizes the agent to promptly service tasks while minimizing the average wait time for all tasks, which is achieved by assigning a decaying reward of $2 \cdot e^{\left(1-\frac{\text{wait\_times}_j}{\text{max\_wait\_time}}\right)}$, where $j$ denotes the serviced task. Here, an exponentially

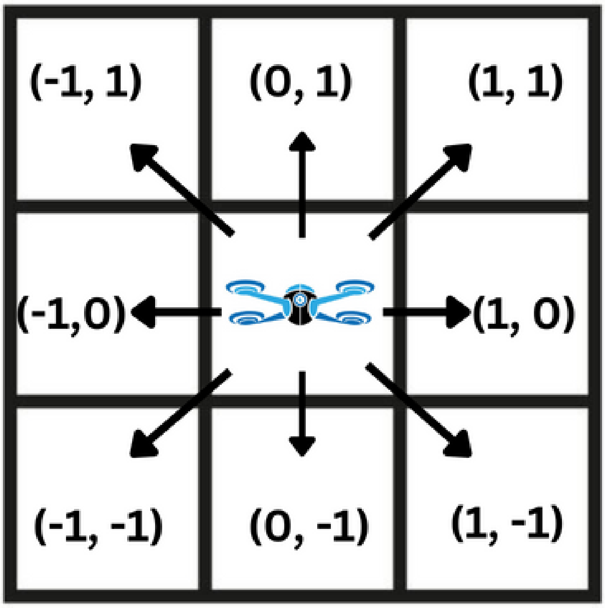


Figure 4.2: The agent is enabled with $8$ discrete actions at any given time step. By selecting an action $a_t \in \mathcal{A}$ at timestep $t$, the agent can traverse to any of its adjacent cells within the grid-world environment.

decaying function is used instead of a linearly decaying function to provide more incentives for completing unserved tasks given a maximum task wait time threshold. In the experimental setup, we do not employ task prioritization, and the cumulative reward maximizes when more tasks are serviced promptly and penalized when unnecessary wait occurs. For consistency on the rewards scaling, we set $\texttt{max\_wait\_time} = 30$ in our experiments. The relation between the task wait time and the reward value is visualized in Fig. 4.3.

The terminal states are those where all $n$ tasks are serviced. An episode is considered truncated when the time limit is reached, i.e., $p = 1$, and the environment parameters are reset to facilitate subsequent training. The reward signal value spans a range of $[-3.5, 74.4]$. Upon reaching a terminal state, the agent receives a reward of $+20$. However, when an episode is truncated, we implement a penalty in the form of a reward signal as follows:

$$r_{\text{trunc}} = \begin{cases} -\dfrac{\text{(total\_tasks - tasks\_serviced)}}{\text{total\_tasks}} & \text{if tasks\_serviced} > 0 \\ -1 & \text{otherwise} \end{cases}$$

This truncation reward signal imposes a penalty proportional to the ratio of unserved tasks to the total number of tasks when tasks have been serviced during the episode. Otherwise, a fixed penalty of $-1$ is applied.

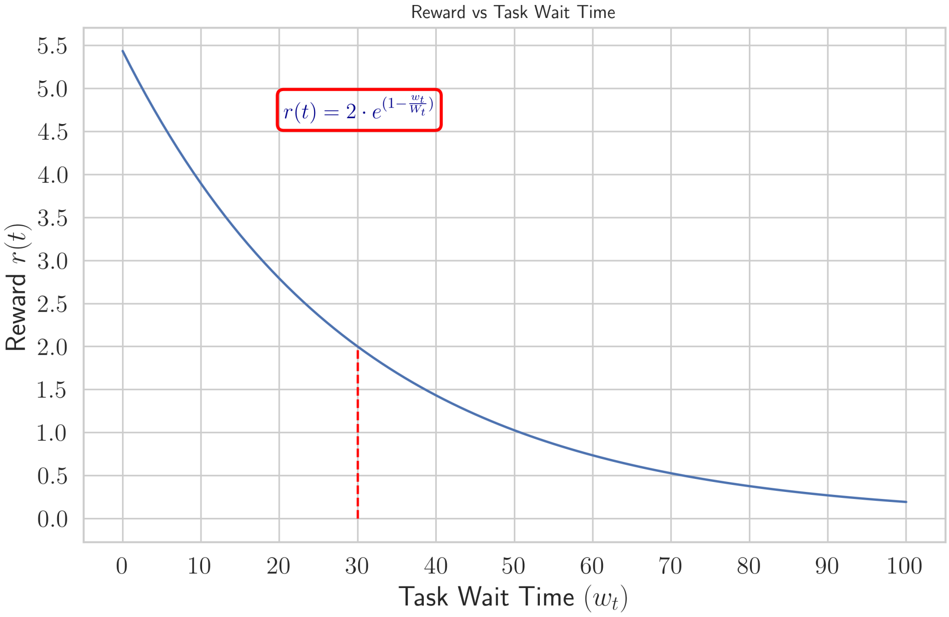


Figure 4.3: The exponentially decaying reward as a function of the task wait times. The max wait time is capped at $W_t = 30$, corresponding to a reward of $+2$. The Deep-RL agent thrives on maximizing its reward by efficiently servicing tasks while minimizing task wait times. In the figure, $w_t$ denotes the wait time of a task.

### 4.2.4 Parallel Training in Vectorized Environment

The intrinsic stochasticity inherent to the dynamics of the DVR environment during training poses a considerable challenge for Deep-RL algorithms. This stochasticity not only hinders the learning process but also necessitates an extensive number of training samples, making the learning process sample inefficient. A vectorized environment was employed in response to these challenges, facilitating parallel processing during training. Vectorized environments play a pivotal role in enhancing the sample efficiency of Deep-RL training by enabling the collection of a larger volume of data within the same timeframe. It is achieved by simultaneously simulating multiple environments in parallel, facilitating numerous interactions with the environment. Vectorized environments also contribute to the stability and reliability of training processes by mitigating the variance of training data. By conducting parallel simulations, vectorized environments effectively smooth out the random fluctuations inherent to individual environments. Additionally, the parallel processing capability of multiple environments optimizes the utilization of hardware resources such as CPUs and GPUs. Consequently, using vectorized environments reduces the training time required for Deep-RL algorithms, enhancing overall efficiency and effectiveness.

The following section explores the experimental setup and results of implementing PPO in vectorized environments in our Deep-RL framework to address the DVR.

# Chapter 5

# Experiments and Results

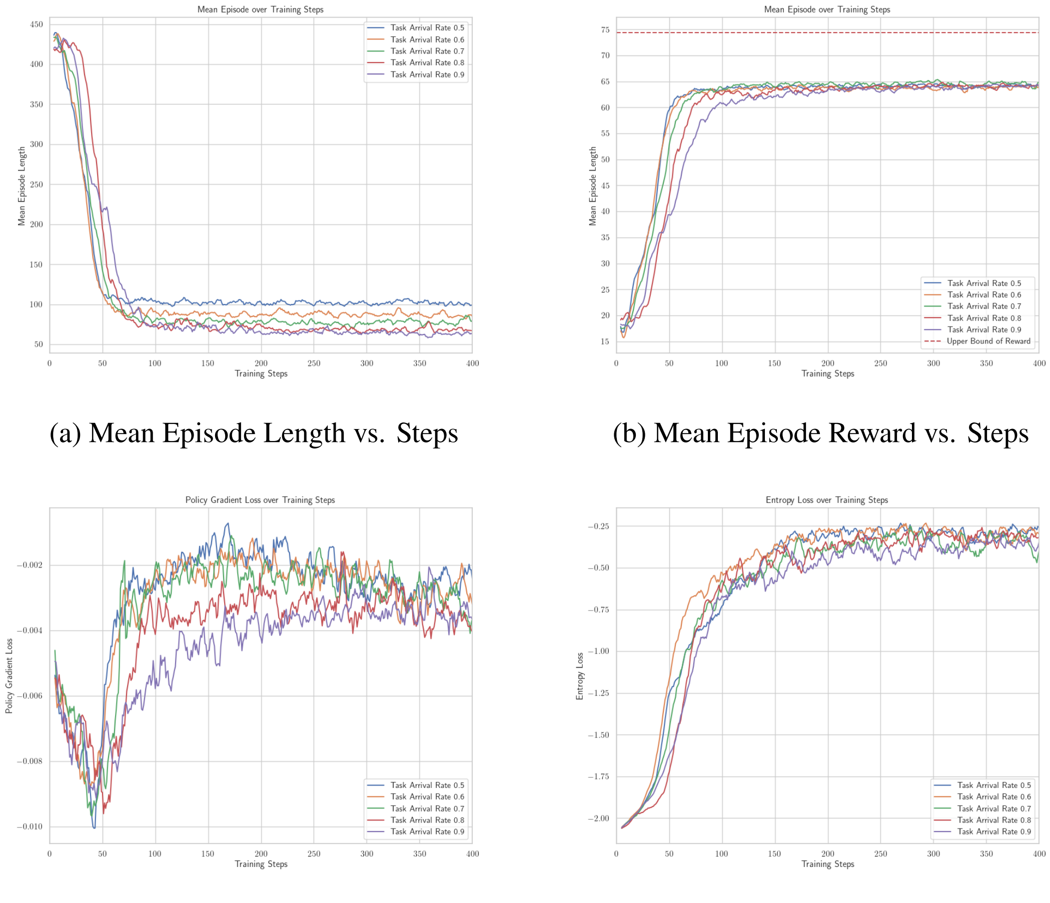
## 5.1 Experimental Setup

As subsection 4.2.2 mentioned, we implemented a grid world environment for the DVR. The environment is a 2-dimensional square grid of size $10 \times 10$. The agent is initially placed in the middle of the environment. Leveraging the `NumPy` library [52], specifically the `numpy.exponential(1/lam)` function, we generate inter-arrival times for tasks following a Poisson process, where `lam` denotes the task arrival rate ($\lambda$). These task arrivals are independent and identically distributed (i.e., i.i.d.), facilitating the modeling of the load factor ($\rho$) for DVR. For training the Deep-RL model, we employ the Proximal Policy Optimization (PPO) algorithm from the Stable Baselines 3 library [53]. Initially, we generate a sequence of 10 tasks for each episode within timesteps $t = [0, 1, 2, \ldots, 100]$, and each configuration is trained for a maximum of 500 timesteps. This extended time limit is deliberately chosen to enable the agent to explore and interact with the environment thoroughly. Episodes conclude upon reaching the terminal state or exceeding the maximum time limit. The Deep-RL agent receives feedback in the form of rewards, as defined in subsection 4.2.3.

We used a `NVIDIA RTX A4000` GPU with 16 GB memory to train Deep-RL models across various task arrival rates for 1 million timesteps. Additionally, we employ 100 task configurations, each comprising a list of 10 tasks, as a test set to evaluate the Deep-RL approach against two greedy heuristic algorithms based on average task waiting times.

## 5.2 Results

In RL, the agent's objective is to learn an optimal policy that maximizes the cumulative rewards over time. However, finding the optimal policy is challenging in complex environments with large state spaces. PPO use policy entropy to tackle this challenge, which encourages exploration and helps prevent the agent from getting stuck in a suboptimal policy. As we progress in the training steps, a gradual increase in rewards over time is observed, indicating the learning curve for the Deep-RL model. In the plots, each training step depicts $2500$ learning steps or actions in parallel environments.



(a) Mean Episode Length vs. Steps

(b) Mean Episode Reward vs. Steps

(c) Policy Gradient Loss vs. Steps

(d) Policy Entropy Loss vs. Steps

Figure 5.1: Comparison of the Deep-RL training results for five different task arrival rates $\lambda \in [0.5, 0.6, 0.7, 0.8, 0.9]$

As the Deep-RL agent becomes more adept at navigating its environment, the average rewards earned per episode steadily increase and eventually stabilize, as shown in Fig. 5.1. In Figure 5.1 (a), we observe that the length of episode has decreased and reached a plateau of around 100. Additionally, Figure 5.1 (b) demonstrates the agent's improving performance over time. Figures 5.1 (c) and (d) indicate that the Deep-RL policy is gaining confidence in the unpredictable nature of the environment, as evidenced by the gradual decrease in policy gradient loss and increase of the policy entropy. However, there are fluctuations in the metrics due to the high stochastic nature of the DVR environment.

## 5.3 Comparison of Deep-RL with Baselines

To assess the efficacy of the Deep-RL algorithm, We test the model for $100$ tasks' list configuration. We compare the Deep-RL approach with two greedy heuristic algorithms denoted as the `baselines` based on the average waiting times of the task for the load factor $\rho \in [0.5, 0.6, 0.7, 0.8, 0.9]$. The figure 5.2 compares the effectiveness of our approach in reducing task wait times to that of the greedy `baselines`.

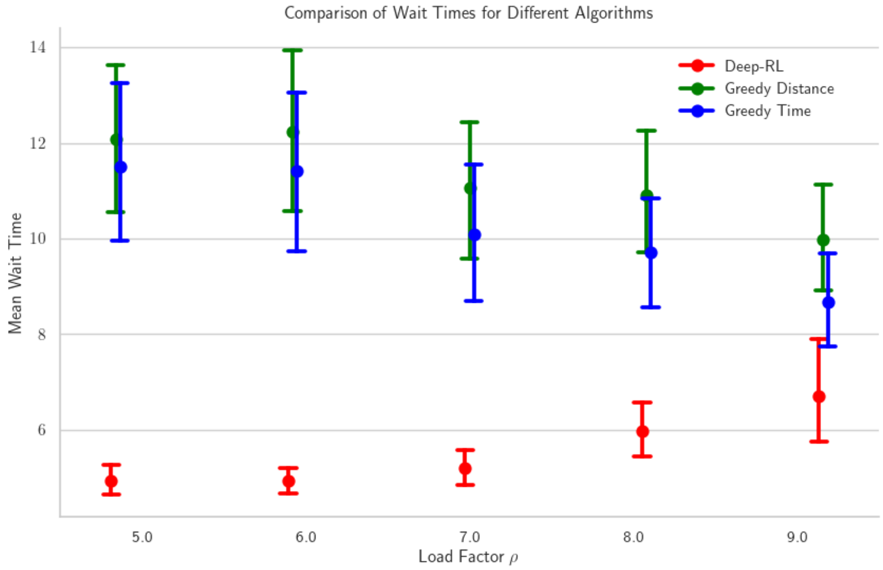


Figure 5.2: Comparison of the Deep-RL approach with two greedy heuristics. The Deep-RL achieves the best performance in comparison to the greedy `baselines`.

In the greedy times approach (GTA), the agent maintains a queue of unserved tasks and selects the one to service that has arrived before the others. And, if there are no unserved tasks in the environment, the agent stays where it serviced the last task. Similarly, for the second greedy distance approach (GDA), the agent maintains a queue of unserved tasks, selects the one to service near the agent's position, and stays there until a new task arrives in the environment. Comparison between these approaches is conducted based on their respective improvements in minimizing the average waiting time for tasks.

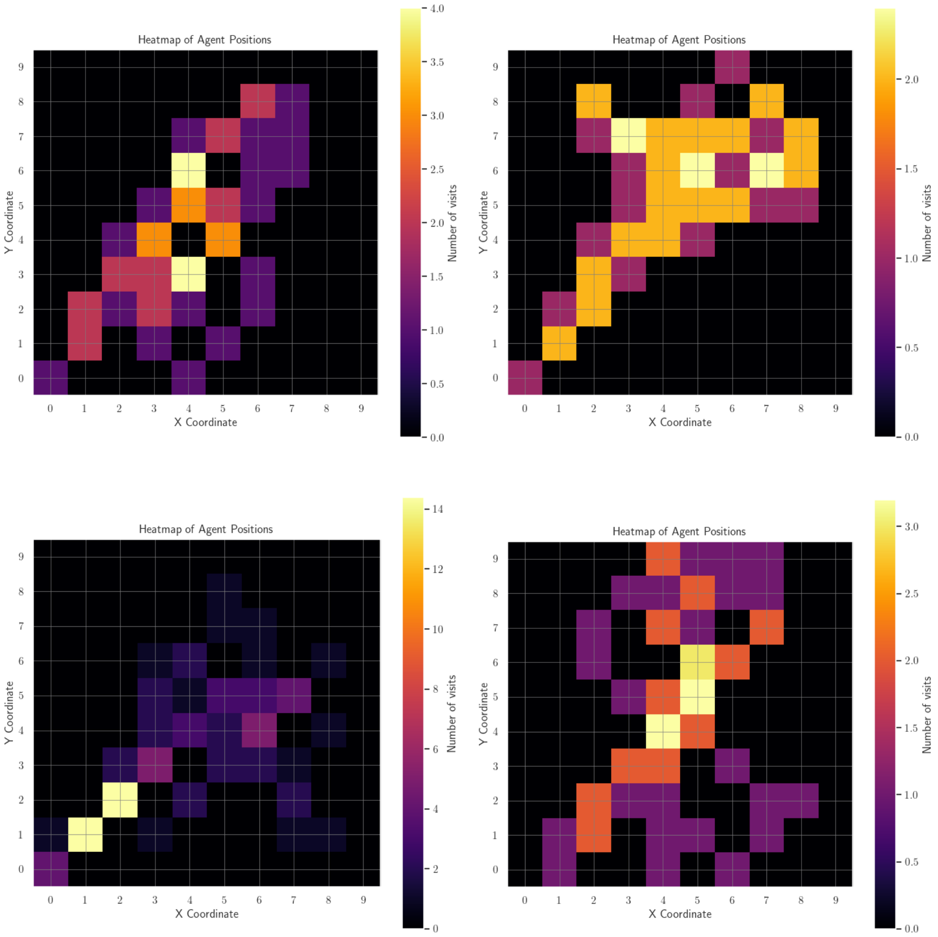


Figure 5.3: A heatmap illustration of the spatial distribution of agent movements over time. Darker regions indicate areas where agents have spent more time, while lighter regions represent less frequented areas. The heatmap provides insight into the navigation patterns and exploration behavior of the agents within the environment.

In an experimental setting involving 100 iterations, each corresponding to the testing of $10$ configuration of tasks, the mean wait time observed for the Deep-RL approach is $5.55$, while for the GDA, it is $11.25$. For the GTA, it is $10.27$. Consequently, the proposed Deep-RL approach demonstrates a noteworthy improvement of $45.98\%$ and $50.66\%$ in minimizing task waiting times compared to the GTA and GDA strategies.

Deep-RL agents learn to optimize the objective through repeated interactions with the environment over millions of time steps. Over time, it acquires the ability to navigate to the center of the environment when no tasks are present. This behavior is depicted in Fig. 5.3, which illustrates a heatmap showing the movement of the agents.

# Chapter 6

# Conclusion & Future Works

The thesis presents a novel Deep Reinforcement Learning (Deep-RL) framework specifically designed to minimize task wait time for dynamic vehicle routing with stochastic task arrivals. The framework is model-free and online, which makes it highly adaptable to various real-world scenarios. The proposed solution has demonstrated significant efficacy when compared to baseline greedy algorithms, leveraging the capabilities of Deep-RL to optimize task wait time for single-agent dynamic vehicle routing.

Our approach has been tested across various load factor configurations, and the results have been promising, with the proposed solution yielding at least 46% more efficient solutions on average. However, it is worth noting that there are limitations to the efficacy of the approach. In scenarios with a high load factor, where there are many targets, the RL approach performs comparably to the greedy algorithms. This is because, in such scenarios, the inter-arrival time between tasks decreases, resulting in more tasks in the system. Therefore, choosing any task to service is as good as making informed decisions.

An extension of this work involves the utilization of a Multi-Agent framework to manage multiple vehicles simultaneously. This approach enables cooperative RL agents to collaborate toward achieving common objectives, resulting in more efficient and effective solutions. Overall, the proposed Deep-RL framework is a promising solution for single-agent dynamic vehicle routing with stochastic task arrivals, and it has the potential to be further extended and adapted to more complex real-world scenarios.

# Bibliography

[1] Jalel Euchi, Adnan Yassine, and Habib Chabchoub. The dynamic vehicle routing problem: Solution with hybrid metaheuristic approach. *Swarm and Evolutionary Computation*, 21:41–53, 2015.

[2] E. Kucharska. Dynamic vehicle routing problem—predictive and unexpected customer availability. *Symmetry*, 11:546, 2019.

[3] V Sreelekshmi and Jyothisha J Nair. Dynamic vehicle routing for solid waste management. In *2017 IEEE Region 10 Symposium (TENSYMP)*, pages 1–5. IEEE, 2017.

[4] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.

[5] Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. *Computers & industrial engineering*, 99:300–313, 2016.

[6] Dimitris J Bertsimas and Garrett Van Ryzin. A stochastic and dynamic vehicle routing problem in the euclidean plane. *Operations Research*, 39(4):601–615, 1991.

[7] Dengkai HOU, Houming FAN, and Xiaoxue REN. Dynamic multi-depot multi-compartment refrigerated vehicle routing problem with multi-path based on real-time traffic information. *Applied Mathematics, Modeling and Computer Simulation: Proceedings of AMMCS 2021*, 20:349, 2022.

[8] Qingchun Hou, Jingwei Yang, Yiqiang Su, Xiaoqing Wang, and Yuming Deng. Generalize learned heuristics to solve large-scale vehicle routing problems in real-time. In *The Eleventh International Conference on Learning Representations*, 2022.

[9] Micha l Okulewicz and Jacek Mańdziuk. A metaheuristic approach to solve dynamic vehicle routing problem in continuous search space. *Swarm and Evolutionary Computation*, 48:44–61, 2019.

[10] Maryam Abdirad, Krishna Krishnan, and Deepak Gupta. A two-stage metaheuristic algorithm for the dynamic vehicle routing problem in industry 4.0 approach. *Journal of Management Analytics*, 8(1):69–83, 2021.

[11] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017.

[12] Francesco Bullo, Emilio Frazzoli, Marco Pavone, Ketan Savla, and Stephen L Smith. Dynamic vehicle routing for robotic systems. *Proceedings of the IEEE*, 99(9):1482–1504, 2011.

[13] Marco Pavone, Emilio Frazzoli, and Francesco Bullo. Adaptive and distributed algorithms for vehicle routing in a stochastic and dynamic environment. *IEEE Transactions on automatic control*, 56(6):1259–1274, 2010.

[14] Nahid Parvez Farazi, Tanvir Ahamed, Limon Barua, and Bo Zou. Deep reinforcement learning and transportation research: A comprehensive review. *arXiv e-prints*, pages arXiv–2010, 2020.

[15] Nahid Parvez Farazi, Bo Zou, Tanvir Ahamed, and Limon Barua. Deep reinforcement learning in transportation research: A review. *Transportation research interdisciplinary perspectives*, 11:100425, 2021.

[16] 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.

[17] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[18] OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemys law Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning, 2019.

[19] Sharada Mohanty, Erik Nygren, Florian Laurent, Manuel Schneider, Christian Scheller, Nilabha Bhattacharya, Jeremy Watson, Adrian Egli, Christian Eichenberger, Christian Baumberger, Gereon Vienken, Irene Sturm, Guillaume Sartoretti, and Giacomo Spigler. Flatland-rl : Multi-agent reinforcement learning on trains, 2020.

[20] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft ii: A new challenge for reinforcement learning, 2017.

[21] Bo Peng, Jiahai Wang, and Zizhen Zhang. A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. In *Artificial Intelligence Algorithms and Applications: 11th International Symposium, ISICA 2019, Guangzhou, China, November 16–17, 2019, Revised Selected Papers 11*, pages 636–650. Springer, 2020.

[22] Simone Foa, Corrado Coppola, Giorgio Grani, and Laura Palagi. Solving

the vehicle routing problem with deep reinforcement learning. *arXiv preprint arXiv:2208.00202*, 2022.

[23] Jiaqi Guo, Jiancheng Long, Xiaoming Xu, Miao Yu, and Kai Yuan. The vehicle routing problem of intercity ride-sharing between two cities. *Transportation Research Part B: Methodological*, 158:113–139, 2022.

[24] Omer Ozkan and Sezgin Kilic. Uav routing by simulation-based optimization approaches for forest fire risk mitigation. *Annals of Operations Research*, 320(2):937–973, 2023.

[25] David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8):2403–2435, 2007.

[26] Shivam Bajaj and Shaunak D Bopardikar. Dynamic boundary guarding against radially incoming targets. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 4804–4809. IEEE, 2019.

[27] Lejun Zhou, Anke Ye, and Simon Hu. A four-stage heuristic algorithm for solving on-demand meal delivery routing problem. *perspective*, 12(13):14, 2022.

[28] Michael Wilbur, Salah Uddin Kadir, Youngseo Kim, Geoffrey Pettet, Ayan Mukhopadhyay, Philip Pugliese, Samitha Samaranayake, Aron Laszka, and Abhishek Dubey. An online approach to solve the dynamic vehicle routing problem with stochastic trip requests for paratransit services. In *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, pages 147–158. IEEE, 2022.

[29] Ward Whitt. Understanding the efficiency of multi-server service systems. *Management Science*, 38(5):708–723, 1992.

[30] Dimitris J Bertsimas and Garrett Van Ryzin. Stochastic and dynamic vehicle routing with general demand and interarrival time distributions. *Advances in Applied Probability*, 25(4):947–978, 1993.

[31] Harilaos N Psaraftis, Min Wen, and Christos A Kontovas. Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67(1):3–31, 2016.

[32] Marco Pavone, Alessandro Arsie, Emilio Frazzoli, and Francesco Bullo. Distributed algorithms for environment partitioning in mobile robotic networks. *IEEE Transactions on Automatic Control*, 56(8):1834–1848, 2011.

[33] Alexander Botros, Barry Gilhuly, Nils Wilde, Armin Sadeghi, Javier Alonso-Mora, and Stephen L Smith. Optimizing task waiting times in dynamic vehicle routing. *IEEE Robotics and Automation Letters*, 2023.

[34] Marlin W Ulmer. Anticipation versus reactive reoptimization for dynamic vehicle routing with stochastic requests. *Networks*, 73(3):277–291, 2019.

[35] Dan Wang, Hong Zhou, and Ruxin Feng. A two-echelon vehicle routing problem involving electric vehicles with time windows. In *Journal of Physics: Conference Series*, volume 1324, page 012071. IOP Publishing, 2019.

[36] Yingchun Wang, Longfei Zheng, Huaguang Zhang, and Wei Xing Zheng. Fuzzy observer-based repetitive tracking control for nonlinear systems. *IEEE Transactions on Fuzzy Systems*, 28(10):2401–2415, 2019.

[37] Waldy Joe and Hoong Chuin Lau. Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers. In *Proceedings of the international conference on automated planning and scheduling*, volume 30, pages 394–402, 2020.

[38] Ahmad Bdeir, Simon Boeder, Tim Dernedde, Kirill Tkachuk, Jonas K Falkner, and Lars Schmidt-Thieme. Rp-dqn: An application of q-learning to vehicle routing problems. In *KI 2021: Advances in Artificial Intelligence: 44th German Conference on AI, Virtual Event, September 27–October 1, 2021, Proceedings 44*, pages 3–16. Springer, 2021.

[39] Weixu Pan and Shi Qiang Liu. Deep reinforcement learning for the dynamic and uncertain vehicle routing problem. *Applied Intelligence*, 53(1):405–422, 2023.

[40] Marlin W Ulmer, Justin C Goodson, Dirk C Mattfeld, and Barrett W Thomas. On modeling stochastic dynamic vehicle routing problems. *EURO Journal on Transportation and Logistics*, 9(2):100008, 2020.

[41] William Uther. *Markov Decision Processes*, pages 642–646. Springer US, Boston, MA, 2010.

[42] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[43] 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.

[44] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[45] 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.

[46] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[47] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[48] 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.

[49] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on ppo and trpo, 2020.

[50] Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. Time limits in reinforcement learning. In *International Conference on Machine Learning*, pages 4045–4054. PMLR, 2018.

[51] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[52] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020.

[53] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.