FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Last-Mile Delivery Problem Representation in Reinforcement Learning

**Clara Alves Martins**

Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Rosaldo J. F. Rossetti

Second Supervisor: Zafeiris Kokkinogenis

July 28, 2023

# Last-Mile Delivery Problem Representation in Reinforcement Learning

**Clara Alves Martins**

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. Ana Paula Cunha da Rocha
External Examiner: Prof. Alberto Fernández Gil
Supervisor: Prof. Rosaldo J. F. Rossetti

July 28, 2023

This work is a result of project DynamiCITY: Fostering Dynamic Adaptation of Smart Cities to Cope with Crises and Disruptions, with reference NORTE-01-0145-FEDER-000073, supported by Norte Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, through the European Regional Development Fund (ERDF).

Cofinanciado por:

# Abstract

The last-mile delivery problem refers to the final phase of the delivery process, the item's trip between the last warehouse and the customer's house. Due to its substantial cost, it makes a great candidate for optimisation. Besides, it embodies many similarities with the well-known Vehicle Routing Problem.

Even though this problem is commonly solved using optimisation-driven techniques, Reinforcement Learning has achieved promising results. Additionally, Graph Deep Reinforcement Learning allows the application of Reinforcement on graph-structured data. It can handle variable-sized graphs and much more complex environments at the cost of more interaction samples. Since getting those extra samples might be costly, Transfer Learning aims to increase sample efficiency by reusing knowledge from previously learnt tasks.

In this project, we focus on the representation of a last-mile delivery problem. We organise a taxonomy for solving the Vehicle Routing Problem using Reinforcement Learning. We utilise two real-world datasets to explore the effects of different node features and reward functions on the Reinforcement Learning agents' performance. On top of that, we devise an empirical study designed to ascertain the effects of Transfer Learning when changing the environment or reward function.

We emphasise the use of masking mechanisms to avoid searching invalid actions. Additionally, we demonstrate that Transfer Learning is highly dependent on the quality of the source model. Finally, we proved that Transfer Learning can be successful when changing Vehicle Routing Problem variants and reward functions.

**Keywords**: Reinforcement Learning, Transfer Learning, Last-Mile Delivery, Vehicle Routing Problem, Representation Problem

# Resumo

O problema da entrega da última milha refere-se à fase final do processo de entregas, a viagem da encomenda entre o último armazém e a casa do cliente. Devido ao seu custo substancial, é um ótimo candidato para otimização. Além disso, apresenta muitas semelhanças com o conhecido Problema de Planeamento de Rotas.

Embora este problema seja geralmente resolvido com técnicas de otimização, a Aprendizagem por Reforço tem alcançado resultados promissores. Além disso, a Aprendizagem por Reforço Profunda em Grafos permite a aplicação do Reforço em dados estruturados em grafos. Pode lidar com grafos de tamanho variável e ambientes muito mais complexos à custa de mais amostras. Uma vez que a obtenção dessas amostras adicionais pode ser muito dispendiosa, a Aprendizagem por Transferência visa aumentar a eficiência das amostras através da reutilização de conhecimentos provenientes de tarefas previamente exploradas.

Neste projeto, concentámo-nos na representação de um problema de entrega da última milha. Organizamos uma taxonomia para resolver o Problema de Planeamento de Rotas utilizando Aprendizagem por Reforço. Utilizamos dois conjuntos de dados do mundo real para explorar os efeitos de diferentes características dos vértices do grafo e funções de recompensa no desempenho dos agentes de Aprendizagem por Reforço. Além disso, elaboramos um estudo empírico destinado a verificar os efeitos da Aprendizagem por Transferência quando se altera o ambiente ou a função de recompensa.

Salientamos a utilização de uma máscara para evitar a procura de ações inválidas. Além disso, demonstrámos que a Aprendizagem por Transferência é altamente dependente da qualidade do modelo de origem. Finalmente, provámos que a Aprendizagem por Transferência pode ser bem sucedida quando se alteram as variantes do Problema de Planeamento de Rotas e as funções de recompensa.

**Keywords**: Aprendizagem por Reforço, Aprendizagem por Transferência, Entrega da Última Milha, Problema de Planeamento de Rotas, Problema de Representação

# Acknowledgments

First and foremost, I would like to thank my supervisor, Rosaldo Rossetti, for his invaluable guidance and patience throughout the entire process. I am grateful for his trust and for helping me select such a relevant and challenging topic. Also to my co supervisor, Zafeiris Kokkinogenis, thanks for being available to discuss ideas, provide feedback and encourage me to always be looking for more.

I would like to express my sincere gratitude to all the individuals and institutions who made this research work possible. This work was part of the project *DynamiCITY: Fostering Dynamic Adaptation of Smart Cities to Cope with Crises and Disruptions*. I am thankful for the time I spent learning with other participants and the insightful conversations I had as a result of this collaboration.

I would also like to extend my thanks to the Faculty of Engineering at the University of Porto (FEUP) for providing essential materials and information for my research. The wide range of resources and infrastructure offered by FEUP was crucial to the development of this study. Furthermore, I would like to express my gratitude to the Laboratory of Artificial Intelligence and Computer Science (LIACC) for providing me with specialized support in the theme of my work.

I cannot fail to acknowledge the support provided by my fellow Master's friends and colleagues who shared their ideas, knowledge, and experiences throughout the last five years. The collaboration and knowledge exchange with them was precious to the advancement of my research. Specially, a heartfelt thanks to Félix and Daniel for listening to me talk about my ideas and helping me review this work.

I also want to express my gratitude to my family for their unwavering support throughout this journey. Their words of encouragement and love were a constant source of motivation. I am forever grateful for your love.

Finally, I would like to thank all those who, in one way or another, contributed to the completion of this project, even if not mentioned here. Your support and encouragement were instrumental to the success of this work.

Clara Martins

# Contents

# List of Figures

# List of Tables

# Listings

# Abbreviations and Acronyms

| | |
|---|---|
| A2C | Actor-Critic |
| A3C | Asynchronous Advantage Actor-Critic |
| C51 | Categorical 51-Atom Deep Q-Networks |
| $CO_2$ | Carbon Dioxide |
| CSP | Covering Salesman Problem |
| CVRP | Capacitated Vehicle Routing Problem |
| CVRPSTW | Capacitated Vehicle Routing Problem with Soft Time Windows |
| CVRPTW | Capacitated Vehicle Routing Problem with Time Windows |
| D3QN | Double Deep Q-Learning Network |
| DDPG | Deep Deterministic Policy Gradient |
| DQN | Deep Q-Networks |
| DRL | Deep Reinforcement Learning |
| DVRP | Dynamic Vehicle Routing Problem |
| EVRP | Electric Vehicle Routing Problem |
| GDRL | Graph Deep Reinforcement Learning |
| GHG | Greenhouse Gas |
| GNN | Graph Neural Networks |
| GVRP | Green Vehicle Routing Problem |
| HCVRP | Heterogeneous Capacitated Vehicle Routing Problem |
| HER | Hindsight Experience Replay |
| I2A | Imagination-Augmented Agents |
| LoggiBUD | Loggi Benchmark for Urban Deliveries |
| MARL | Multi-Agent Reinforcement Learning |
| MBMF | Model-Based RL with Model-Free Fine-Tuning |
| MBVE | Model-Based Value Expansion |
| MDP | Markov Decision Process |
| NN | Neural Networks |
| PCTSP | Prize Collecting Traveling Salesman Problem |
| PDP | Pickup and Delivery Problem |
| PPO | Proximal Policy Optimization |
| QR-DQN | Quantile Regression Deep Q-Networks |
| REINFORCE | REward Increment = Nonnegative Factor x Offset Reinforcement x Characteristic Eligibility |
| RL | Reinforcement Learning |
| RNN | Recurrent Neural Network |
| SAC | Soft Actor-Critic |
| SDVRP | Split Delivery Vehicle Routing Problem |
| SPCTSP | Stochastic Prize Collecting Traveling Salesman Problem |

| | |
|---|---|
| SVRP | Stochastic Vehicle Routing Problem |
| TD3 | Twin Delayed DDPG |
| TL | Transfer Learning |
| TRPO | Trust Region Policy Optimization |
| TSP | Traveling Salesman Problem |
| VRP | Vehicle Routing Problem |
| VRPDO | Vehicle Routing Problem with Delivery Options |
| VRPPD | Vehicle Routing Problem with Pickup and Delivery |
| VRPSSR | Vehicle Routing Problem with Stochastic Service Requests |
| VRPSTW | Vehicle Routing Problem with Soft Time Windows |
| VRPTW | Vehicle Routing Problem with Time Windows |

# Chapter 1

# Introduction

This chapter introduces the topic of this work. It starts by introducing the context and motivation for this project, followed by how we define the problem. Next, it specifies what we aim to accomplish and how we plan to achieve our goal. In the end, it outlines the rest of this document.

## 1.1 Context and Motivation

The last-mile delivery problem corresponds to the transportation of each order from the last warehouse to the corresponding customer. It has become a very prominent problem for delivery companies due to the proliferation of e-commerce and free shipping [23]. Since the majority of the population lives in urban areas [41], most of these deliveries are scattered throughout the cities. Due to its high cost compared with the rest of the trip [23], and its similarities with variants of the Vehicle Routing Problem (VRP), it is a great candidate for optimisation. In real-world situations, it is possible that, in addition to minimising the costs (either the total distance travelled or time used), other factors might need to be considered as well. Besides, with ecological solutions gaining popularity and the possibility of delivery orders with the help of crewless vehicles (e.g., robots or drones), new optimisation needs come to light.

Reinforcement Learning (RL) is a machine learning technique focused on learning from interactions with the environment. It attempts to maximise the total received reward over a sequence of actions. Traditional Reinforcement Learning methods mainly address scenarios where the states and actions can be represented in a tabular form. However, due to their complexity, this approach makes many real-world problems infeasible. Deep Reinforcement Learning (DRL), which combines Reinforcement Learning with Deep Learning, has recently gained popularity.

On the one hand, Deep Reinforcement Learning has a powerful representation ability and can deal with more complex tasks much closer to real-world scenarios [130]. Additionally, Graph Deep Reinforcement Learning (GDRL) combines the learning power of Deep Reinforcement Learning with the versatility and ability to handle graphs from Graph Neural Networks (GNN).

Moreover, it is not limited to fixed-sized inputs, enabling the use of different-sized graphs. Since the Vehicle Routing Problem can be naturally formulated in graph form, solving it using Graph Deep Reinforcement Learning looks promising. On the other hand, it suffers from the curse of dimensionality [66], and its training may require many interaction samples.

In a real-world scenario, getting those samples may be costly or even prohibitive. To mitigate this adversity, one can use Transfer Learning (TL) alongside Reinforcement Learning. Transfer Learning is a method of reusing knowledge gained from solving one problem to solve a related but different problem. It aims to reduce the number of required samples by increasing sample efficiency and can be especially useful when the new task has a limited amount of data or resources. However, there is still uncertainty about which representations, or parts of them, better influence Transfer Learning techniques. Transfer Learning methods depend highly on the knowledge transferred since it determines how well one can apply prior knowledge to the new task [160]. Therefore, the problem of representation is a key factor in the success of both Reinforcement Learning and Transfer Learning.

## 1.2 Problem Definition

Reinforcement Learning is a powerful and generalisable methodology. Unfortunately, it encounters multiple challenges when applied to real-world problems. These challenges include high dimensionality of state and action spaces, limited data, training offline from previously collected data, and multi-objective reward functions [26, 25].

The representation problem in Reinforcement Learning refers to representing the environment in a way that will ease the agent's learning and decision-making process. "A machine learning model can't directly see, hear, or sense input examples" [21]. Instead, it relies on its representation of the environment, both the state's and its internal representation. Without a suitable representation, it will not capture the important characteristics of the environment. Additionally, it will impair the model's learning and limit its potential.

In short, we wish to find out how to represent the problem in a way that will benefit the agent's performance. Firstly, by scrutinising the effects of representation and reward function in the agent's learning and behaviour. Then, by inspecting the impact of knowledge transfer.

## 1.3 Objectives

The main goal of this research is to gain insights into how the representation problem of Reinforcement Learning affects the agent's performance. By leveraging Transfer Learning to transfer knowledge about these representations, we aim to improve the performance and reduce the number of samples necessary to train the Reinforcement Learning agents. For this purpose, we will devise an empirical experiment to test these representations in a Vehicle Routing Problem domain.

We seek to understand the impact of different representations and reward functions in the learning and performance of Reinforcement Learning agents. Additionally, we aspire to shed some light on the effects of utilising knowledge from previously learnt tasks.

In a broad sense, possible questions that may arise in the pursuit of these goals are:

- How does the choice of state representation affect the training and performance of a Reinforcement Learning agent?
- How does the selection of reward function affect the behaviour of Reinforcement Learning agents?
- How can we guarantee that hard constraints are not violated during the training and testing phases?
- Can the knowledge gained from training in a specific environment be transferred to another environment?
- Can we benefit from transferring knowledge of an agent trained under a different reward function?

## 1.4 Document Structure

This document is divided into six chapters, beginning with this introduction. Chapter 2 provides the necessary background information and relevant concepts, followed by a discussion of the application domain and related work in Chapter 3. The environment and performance metrics are outlined in Chapter 4, and the experiments and their results are analysed in Chapter 5. Finally, the dissertation in Chapter 6 concludes with our contributions and possible avenues for future work.

# Chapter 2

# Background

This chapter defines the concepts and principles required to fully comprehend the project. It includes relevant information and definitions of Reinforcement Learning, Graph Deep Reinforcement Learning, and its relevant algorithms and Neural Network layers. Additionally, it covers the concept of Transfer Learning and the importance of representation in Reinforcement Learning and Transfer Learning.

## 2.1 Reinforcement Learning

Machine Learning paradigms are usually divided into three categories: Supervised, Unsupervised, and Reinforcement Learning (RL). Even though there are similarities between the three, RL distinguishes itself by its unique characteristics. Like Supervised Learning, RL attempts to find a mapping between states (inputs) and actions (outputs). However, unlike Supervised Learning, RL is not about finding the best immediate reward but the cumulative reward after performing all the actions. Like Unsupervised Learning, RL does not require a labelled dataset. However, unlike Unsupervised Learning, RL does not intend to find a pattern in the data, only the actions that lead to greater rewards.

There are also similarities between RL and other optimisation methods due to RL's objective of maximising the received reward. Even though its primary goal is optimisation, it is quite distinct from other optimisation methods, such as Ant Colony Optimisation [24], Simulated Annealing [54], Tabu Search [40], Exact Methods [118], Constraint Programming [100], and others [59, 14]. A major drawback of these methods is the time and computation required to obtain an instance's result. Whereas RL requires significant computation during its training phase, afterwards, it can yield an answer to a new instance almost instantaneously.

RL learns from interactions with its environment. It is sequential and considers the long-term effect of its actions. RL involves a trial-and-search strategy throughout the possible environment's states to discover the sequences of actions that allow it to reach the highest total reward [111].

For that, it uses feedback from the environment to update its behaviour. This feedback is limited, and there is no clear and explicit indication of which action leads to what reward. Because of this uncertainty, RL usually faces the exploration-exploitation trade-off, where it must choose between a new action to learn more about the environment (exploration) and the best-known action to maximise the reward (exploitation) [111].

RL is usually associated with the concept of agents. One can identify two main elements in a reinforcement-learning system: the agent and the environment.

- The *agent* is the learning system.
- The *environment* is the world around the agent and in which the agent operates.

In addition, there are four subelements of a reinforcement-learning system: the policy, the reward signal, the value function, and the model [111].

- The *policy* defines the agent's behaviour at a specific time.
- The *reward signal* defines the agent's goal.
- The *value function* specifies what a good action in the long run (not only immediately) is.
- The *model* is optional and (mainly) used for planning since it allows the agent to anticipate the behaviour of the environment after performing a specific action.

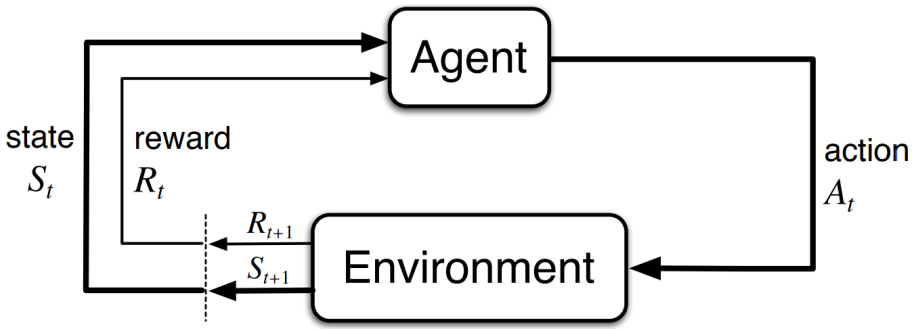


Figure 2.1: Interaction between the Agent and the Environment in a Markov Decision Process [111]

The sequential interaction between the agent and the environment can be modelled as a Markov Decision Process (MDP) [9], as showcased in Figure 2.1. An MDP is a mathematical framework to deal with stochastic control optimisation problems [111] by modelling decision-making processes where the future state is partly random and partly controlled by the agent's decisions. An MDP can be described using a 5-tuple $\langle S, A, T, R, P \rangle$, where

- $S$: state space, the set of possible states $s$.
- $A$: action space, the set of possible actions at each state.

- $T$: the set of all the steps in which the agent makes a decision.
- $R$: reward function, specifying the expected immediate reward for reaching the state *s'* after performing action $a$ in state $s$.
- $P$: the transition probability function, $p(s'|a,s)$, defining the conditional probability of reaching state *s'* after executing action $a$ in state $s$.

A process is Markovian if the transition probability function is only affected by the current state and not by previous states or actions [119].

$$P(s_t, a_t, s_{t+1}) = p(s_{t+1}|s_t, a_t) = p(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2}, \dots) \tag{2.1}$$

Even though the learning process of RL makes it very promising, there are still challenges related to its application in real-world scenarios. Simulated environments are abstractions of the real system, which means some real-world details might not be present in the simulation. There could also be some uncontrollable external factors impacting the agents. Additionally, working in a real-world environment limits the exploration of the agent. On top of that, difficulties may also arise from the interaction with other agents. Multi-Agent Reinforcement Learning (MARL) introduces multiple coexisting RL agents in the same environment. These agents usually need to develop strategies to cooperate or compete with others to fulfil their goals. In multi-agent environments, each time an agent makes an action, it modifies the environment in which the other agents operate, which raises new challenges. In summary, the challenges faced by RL agents in real-world settings include the following [26, 25]:

- Having a limited number of samples, either because of their cost or inability to obtain them on the real system.
- Having unknown and / or large delays on its sensors, actuators, or rewards (i.e., sparse rewards).
- Need to act on a highly dimensional or continuous state and action spaces.
- Never or seldom violating system constraints, either operational or safety.
- Having a non-stationary or stochastic environment causing the tasks to be only partially observable.
- Having multi-objective, risk-sensitive, unspecified, or poorly specified reward functions.
- Providing actions quickly and in a timely manner, especially on systems requiring low latencies.
- Training offline from logs of an external behaviour policy.
- Having to provide system operators with explainable and interpretable policies.

- Need to adapt and generalise between tasks [144].

- Need to avoid models with a huge number of parameters [50].

- Need to speed up the agent's learning process [116].

- Need to transfer knowledge between agents with different internal representations [115, 114].

According to how the agent learns, RL algorithms can be divided into Online RL and Offline RL. In Online RL, the agent learns from interacting with the system in real-time. In contrast, in Offline RL, the agent trains from a set of collected experiences (logs) without the need to interact with the environment during training. Additionally, RL can be divided according to whether or not the agent can access or learn a model of the environment. Figure 2.2 contains an overview of RL algorithms based on this classification. Model-based RL algorithms use the model of the environment directly for planning, as it can predict the environment's state and reward after the execution of an action. They can either be given the model, like in AlphaZero [105], or learn it, like in World Models [43], Imagination-Augmented Agents (I2A)[132], Model-Based RL with Model-Free Fine-Tuning (MBMF) [85], or Model-Based Value Expansion (MBVE) [29]. Model-free RL algorithms can be classified based on what they learn. Q-Learning methods, also known as value-based methods, estimate the action value function using a function approximator. This category encompasses algorithms such as Deep Q-Networks (DQN) [84], Categorical 51-Atom DQN (C51) [8], Quantile Regression DQN (QR-DQN) [17], and Hindsight Experience Replay (HER) [4]. In contrast, Policy Optimisation algorithms focus on learning the policy directly and include methods such as Policy Gradient [112], Advantage Actor-Critic (A2C), Asynchronous Advantage Actor-Critic (A3C) [83], Proximal Policy Optimisation (PPO) [104], and Trust Region Policy Optimisation (TRPO) [103]. Additionally, there is a group of algorithms that bridge the gap between the two categories of model-free RL methods, including Deep Deterministic Policy Gradient (DDPG) [66], Twin Delayed DDPG (TD3) [38], and Soft Actor-Critic (SAC) [44].

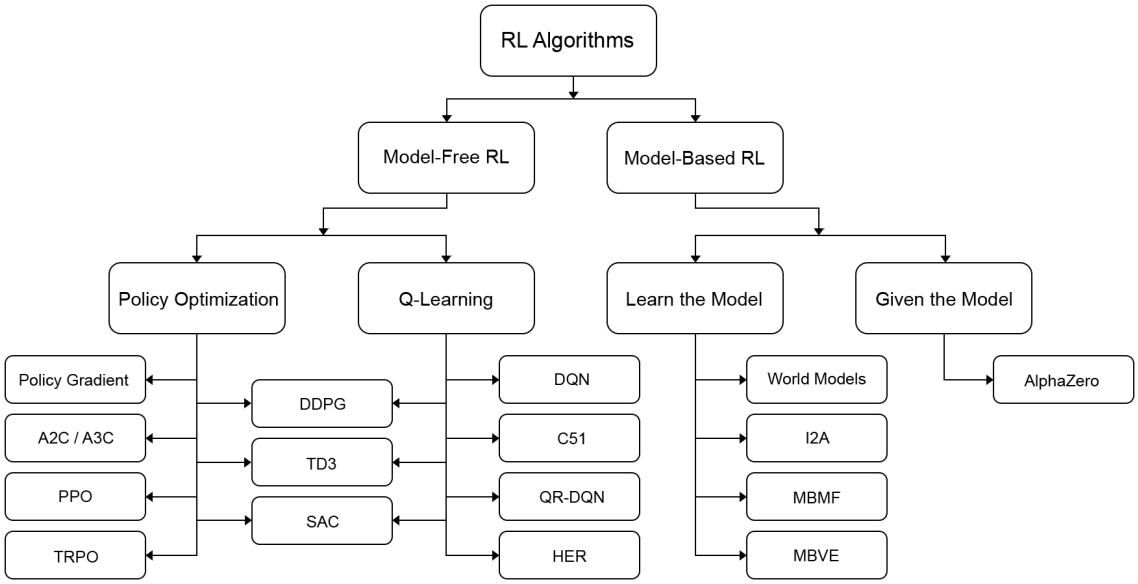


Figure 2.2: Non-exhaustive Taxonomy of Reinforcement Learning Algorithms, adapted from [87].

Note that the Policy Gradient is not an algorithm. Instead, it is a category of algorithms, including, for example, the REINFORCE algorithm [135]. According to [135], REINFORCE is an acronym for "REward Increment = Nonnegative Factor x Offset Reinforcement x Characteristic Eligibility". This algorithm is explored further in Section 2.1.2.

### 2.1.1 Graph Deep Reinforcement Learning

Traditional RL methods mainly address cases where the states and actions can be represented in a tabular form. However, many real-world problems are highly dimensional and are infeasible this way. Consequently, Deep Reinforcement Learning (DRL) has become a trend in dealing with high-dimensional problems. DRL fuses RL with Deep Learning (originally from Supervised Learning), allowing us to handle a much more complex environment at the cost of more interaction samples. It employs Neural Networks (NN) to approximate the value function or policy of the RL agent.

Although Neural Networks allow us to solve more problems, they still present drawbacks. One of these limitations is their fixed input and output sizes. NNs are composed of multiple connected processing layers which successively transform the input [36]. Figure 2.3 represents a Neural Network with two input neurons, a hidden layer with four neurons, and an output layer with a single neuron. Usually, a Neural Network includes one input layer, one output layer, and a variable number of hidden layers. In this architecture, we would always need to input two values and get a single output value. If we wanted to input a third value, we would need to modify the Neural Network's architecture, adding a third input neuron.

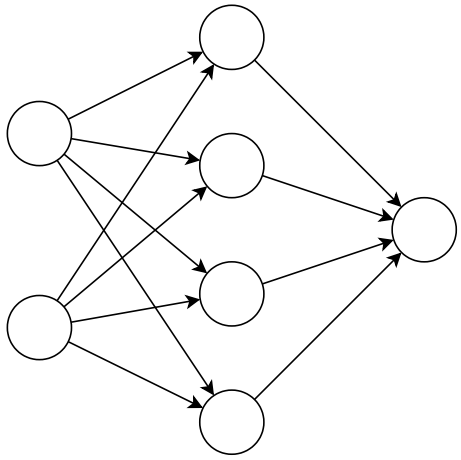

Figure 2.3: Example of a Neural Network with a Single Hidden Layer

Graph Deep Reinforcement Learning (GDRL) blends together Deep Reinforcement Learning with Graph Neural Networks (GNN). On the one hand, it uses the algorithms from DRL. On the other hand, it utilises the GNN's capacity to deal with graphs to take advantage of the environment's graph scheme. GNNs provide a way to define both node and edge features, allowing us to characterise the graph in its entirety. Moreover, they are not dependent on the graph size and do not need alterations when modifying the graph size. This essential factor enables the use of variable-sized inputs under the format of graphs with different sizes.

### 2.1.2 REINFORCE

First presented in 1992 [135], the REINFORCE algorithm has since become a foundational method in the RL field. It is a Monte Carlo Policy Gradient algorithm [111], sharing characteristics with both Monte Carlo and Policy Gradient methods.

In RL, Monte Carlo methods employ techniques based on averaging sample returns. To ensure that the rewards are completely defined and available, they are typically used only for episodic tasks [111].

As presented before in Section 2.1, the Policy Gradient belongs to the class of Policy Optimisation algorithms. These algorithms learn a policy directly and use it to map states to actions. Policy Gradients can learn any parameterised policy as long as it is differentiable with respect to its parameters [111]. They aim to optimise its parameters in order to maximise the expected return. To achieve this, they update their policy through gradient ascent, moving towards a direction that leads to more favourable actions. Additionally, Policy Gradient generates the samples used to update the policy using the policy itself, making them inherently on-policy [96].

In short, the REINFORCE algorithm starts by using its policy to choose the actions to execute (on policy) until it completes an episode. Afterwards, for each step of the episode, it updates its

policy in two steps. First, it calculates the discounted returns using the formula:

$$G_t \leftarrow \sum_{k=t+1}^{T} (\gamma^{k-t-1} R_k) \tag{2.2}$$

, where

- $t$ is the current step.
- $T$ is the total number of steps in the episode.
- $G_t$ is the discounted return at step $t$.
- $\gamma$ is the discount factor.
- $R_k$ is the reward obtained in step $k$.

Then, it updates the policy using the previously calculated discounted returns.

$$\theta \leftarrow \theta + \alpha \gamma^t G_t \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \tag{2.3}$$

, where

- $t$ is the current step.
- $\theta$ is the policy parameters.
- $\alpha$ is the learning rate or step size.
- $\gamma$ is the discount factor.
- $G_t$ is the discounted return at step $t$.
- $\nabla \pi(A_t|S_t, \theta)$ is the gradient of the policy function with respect to the policy parameters. It represents the direction in which the parameters should be updated to increase the likelihood of selecting action $A_t$ in state $S_t$.
- $\pi(A_t|S_t, \theta)$ is the probability of the policy choosing action $A_t$ when in state $S_t$ under the current policy parameterised by $\theta$.

In short, the REINFORCE algorithm can be summarised as follows. $\pi(\cdot|\cdot, \theta)$ represents the policy function.

**Algorithm 1** REINFORCE Algorithm

1: **Input**: a differentiable policy parameterisation $\pi(a|s,\theta)$ and the learning rate $\alpha$
2: Initialise policy parameters $(\theta)$
3: **loop** each episode
4: Generate an episode following $\pi(\cdot|\cdot,\theta)$: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
5: **loop** each step of the episode
6: $G_t \leftarrow \sum_{k=t+1}^{T}(\gamma^{k-t-1}R_k)$
7: $\theta \leftarrow \theta + \alpha\gamma^t G_t \frac{\nabla\pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}$
8: **end loop**
9: **end loop**

### 2.1.3 Graph Convolutional Networks

Graph Convolutional Networks [53] are widely recognised methods for learning graph representations [152]. They take a graph as input, specifically its node and edge features, and generate a new representation for each node [137]. Similarly to other Neural Networks, they are composed of connected layers.

For each layer $k$, $\mathrm{H}^{(k-1)}$ and $\mathrm{H}^{(k)}$ symbolise the input and output node representations, respectively. Additionally, $S$ denotes the normalised adjacency matrix with added self-loops and can be formalised as $\mathrm{S} = \tilde{\mathrm{D}}^{-\frac{1}{2}}\ \tilde{\mathrm{A}}\ \tilde{\mathrm{D}}^{-\frac{1}{2}}$, with $\tilde{\mathrm{D}}$ as the degree matrix of $\tilde{\mathrm{A}}$, $\tilde{\mathrm{A}} = \mathrm{A} + \mathrm{I}$, and $I$ as the identity matrix.

Each GCN layer updates the node features in three steps [137].

1. Apply a local smoothing, propagating the node features and averaging them with the features in their neighbourhood.

$$\bar{\mathrm{H}}^{(k)} \leftarrow \mathrm{S}\ \mathrm{H}^{(k-1)} \tag{2.4}$$

2. Perform a linear transformation using the layer weights ($\Theta^{(k)}$).

$$\bar{\mathrm{H}}^{(k)} \leftarrow \bar{\mathrm{H}}^{(k)}\ \Theta^{(k)} \tag{2.5}$$

3. Apply a nonlinear activation function (e.g., *ReLU*).

$$\mathrm{H}^{(k)} \leftarrow ReLU(\bar{\mathrm{H}}^{(k)}) \tag{2.6}$$

Figure 2.4 showcases the steps of the update. For ease of understanding, each step uses a different colour and line style.

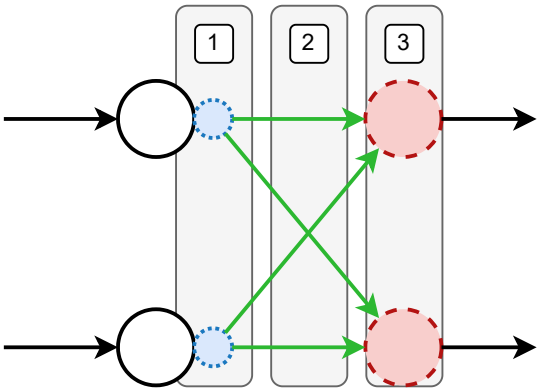


Figure 2.4: Steps for Updating the Feature Vectors on a Graph Convolutional Network Layer

## 2.2 Transfer Learning

Transfer Learning (TL) is a machine learning technique that transfers knowledge from a source task to a target task. It aims to improve the learner's performance or reduce the number of samples needed in a target task by reusing knowledge from previously learned tasks [161]. It is mainly encouraged by the difficulty and cost of acquiring the required data to train the model. Instead of depending on large quantities of data from the specific problem needed to start over and learn how to solve the problem from scratch, one can reuse knowledge from a similar task with more available or easily collectable data. While in an RL problem, the agent is supposed to generalise between instances in the same domain. When applying TL, the agent should be able to generalise beyond the source task domain to the target task domain, which may be different.

One of the most critical factors of the TL procedure is the effect the transferred knowledge has on the target task's learning. Depending on the effect that the transferred knowledge has on the target learner, there are three types of knowledge transfer:

1. *Positive transfer*: the transferred knowledge and previous experience positively influence the target's learning process.

2. *Neutral transfer*: the knowledge transfer does not affect the target's learning process. The learner does not benefit from having previous experience, but that knowledge does not impair their learning.

3. *Negative transfer*: the transferred knowledge has a detrimental effect on the target learner and impairs the learning process. The previous experience inhibits the learner's performance in the target task.

Ideally, we want to ensure positive transfer in all our TL applications. However, so far, it is not clear in the literature precisely what causes the knowledge transfer to be successful or unsuccessful. It is known and understandable that the source and target domains must have something

in common in order to reuse helpful information between the two tasks. If the source and target domains have nothing in common, the knowledge transfer will not contribute to learning how to solve the target task. Despite that, even if the task domains are similar, TL is not guaranteed to have a positive effect since similarities between the domains and tasks can be misleading.

Apart from the effect on the target learner, TL approaches can also be characterised by the number of interactions needed from the target domain, the difference between the domains, the transferred knowledge, and other factors [160].

Based on the number of interactions needed from the target domain, TL approaches can be divided into [160]:

- *Zero-shot* transfer provides ready-to-apply knowledge to the target-domain agent.
- *Few-shot* transfer requires a small number of interactions with the target domain.
- *Sample-efficient* transfer improves the efficiency of each interaction with the target domain.

Based on the similarities and differences between the source and target domains, TL approaches can be classified as follows:

- *Homogeneous*: when the source and target tasks share the same feature space.
- *Heterogeneous*: when that does not happen. These approaches are inherently more complicated since, in addition to distribution adaptation, they also require feature space adaptation [19].

Domain adaptation involves aligning the distributions of the source and the target tasks [99], while feature space adaptation involves aligning their feature spaces [161].

There are multiple forms of knowledge that can be transferred between the source and the target learner. The strategy and framework used for TL depend on the transferred knowledge's representation, granularity, and quality [160]. We can apply TL in RL in multiple forms, including fine-tuning a pre-trained model and transferring the agent's policy or value function.

## 2.3 Representation Problem

Even though the authors of [111] do not address representation, they still consider the representation of states and tasks to affect the learner's performance. As referred to in Section 2.2, TL methods also rely on the underlying knowledge representation. Without the proper information, the learner may suffer considerable impacts on efficiency, accuracy, and transferability.

Problem representation is an essential step for problem-solving [119, 49]. Incorrect or partial information hinders the perception and problem-solving skills, making the problem harder or unsolvable. Similarly, a good representation will ease the understanding of the data [21]. Consequently, it is crucial to consider and define the problem and the necessary data to solve it. In

computer science and artificial intelligence, there is a wide range of possibilities to represent a problem. Two central topics are what information needs to be represented and how that information is structured and organised [119]. It can be done for a variety of reasons, such as to make the information easier to process, to enable the use of certain algorithms or techniques, or to facilitate communication or storage of the data.

Since the model only interacts with the environment via the state's representation and makes decisions based on its internal representation, both these representations significantly influence the learned policy and the transferred knowledge. Additionally, the reward function affects the agent's learning efficiency. By guiding the agent towards the goal, the reward function can affect the learner's performance. A better reward function will lead the agent better and help it learn the optimal policy faster.

# Chapter 3

# Literature Review

Before starting, looking around and reading about how others solved similar problems is essential. Thus, this chapter reviews the existing literature on this project's topic. Firstly, we present the last-mile delivery and the possibilities to abstract it as variants of the Vehicle Routing Problem (VRP). Secondly, we assess how GDRL has been used to solve Vehicle Routing Problems in the literature. Then, we review how TL has been used alongside DRL to solve different VRP variants. Later, we dive into how the representation affects the performance of RL models and how TL can be used to improve this performance. Finally, we summarise the most relevant findings and present a VRP-RL taxonomy for solving the VRP using RL.

## 3.1 Application Domain

The last-mile delivery corresponds to the last step of the delivery process, the item transportation between the warehouse and the customer, and embodies many similarities with the well-known Vehicle Routing Problem. It is a very costly portion of the item's transportation due to the size and dispersal of individual package destinations. These factors make it a great candidate for optimisation.

When optimising this delivery problem, it is typical to abstract it as a variant of the VRP. The VRP is a generalisation of the famous Travelling Salesman Problem (TSP). Just like the TSP, the VRP is a combinatorial optimisation problem. It aims to find the shortest routes for a fleet of vehicles to visit customers at different locations and return to the starting point [22].

Just like the TSP, the VRP possesses many possible researchable variants. The specific variant used depends on the needs and constraints of the problem in question. When optimising a last-mile delivery problem, one could use some of the common VRP variants, such as:

- *Capacitated Vehicle Routing Problem* (CVRP): the vehicles have a specified maximum weight or volume capacity.

- *Vehicle Routing Problem with Time Windows* (VRPTW): the vehicles must visit the delivery locations during the time windows requested by the customers.
- *Capacitated Vehicle Routing Problem with Time Windows* (CVRPTW): incorporates both the previous constraints, the maximum vehicle capacity, and the time windows to deliver the packages.
- *Vehicle Routing Problem with Soft Time Windows* (VRPSTW): if the vehicles visit the delivery locations outside the time windows specified by the customers, they will incur a penalty.
- *Capacitated Vehicle Routing Problem with Soft Time Windows* (CVRPSTW): similar to the CVRPTW, but considering soft time windows, which may be missed at the cost of a penalty.
- *Vehicle Routing Problem with Pickup and Delivery* (VRPPD): the vehicles must transport the item from the pickup points to the respective delivery points [107].
- *Vehicle Routing Problem with Delivery Options* (VRPDO): the vehicle must deliver each item to one of its delivery options. A delivery option corresponds to a location and its associated time window [117].
- *Split Delivery Vehicle Routing Problem* (SDVRP): the customer's demands can be split over multiple routes [56].
- *Heterogeneous Capacitated Vehicle Routing Problem* (HCVRP): the vehicles may have different maximum capacities and speeds [63].
- *Vehicle Routing Problem with Stochastic Service Requests* (VRPSSR): we are unaware of which and when customers will request service, but all potential customers' locations are already known [57].
- *Dynamic Vehicle Routing Problem* (DVRP): the routes are constantly updated as new customer requests arrive.
- *Stochastic Vehicle Routing Problem* (SVRP): as considered in [86], the customer's locations and demands can change.
- *Green Vehicle Routing Problem* [32, 6] (GVRP): weights ecological and environmental concerns.
- *Electric Vehicle Routing Problem* [68] (EVRP): the vehicle has a limited battery which may need to be recharged between customer visits.

Like other combinatorial optimisation problems, the VRP and its variants can be solved using exact or heuristic algorithms [101]. The main difference between these two types of algorithms is the optimality guarantee. Exact methods are guaranteed to find the optimal solution; however, since the problem is NP-hard, they require exponential effort. Heuristic methods are usually faster,

but the solution may not be optimal. Since RL does not have optimality guarantees, it falls under the category of heuristic algorithms.

In real-world problems, it may be necessary to contemplate multiple factors at once or consider multi-objective optimisation. The need to simultaneously optimise multiple conflicting objectives arises from the different optimising needs considered. The optimising needs may include the following:

- Reducing either the distance or time costs.
- Reducing the number of vehicles needed.
- Improving the quality of the trip by considering parking availability, avoiding specific streets, or other similar factors.
- Reducing the amount of pollution produced, by:
  - Improving the trip's energy efficiency.
  - Reducing fuel consumption.
  - Reducing carbon footprint, Greenhouse Gas (GHG), or carbon dioxide ($CO_2$) emissions.

Note that minimising the distance and time costs might already be conflicting if we take into account traffic.

## 3.2 Related Work

Due to the relevance and economic interest of the VRP, it has been extensively researched. Here, we focus on the research relevant to this project. We start by exploring the most relevant topics and research on the topics of DRL, TL, and Representation Learning. Since DRL's introduction, there has been a lot of research on its application for the VRP. Although DRL can handle highly dimensional scenarios and environments, it requires a large number of samples. TL attempts to improve sample efficiency and reduce the number of samples necessary to achieve a good or acceptable performance. Even though there is a more prominent research focus on algorithms and their effect on performance, representation is still crucial to their performance. Afterwards, due to the natural representation of the VRP as a graph, we examine the literature on the GDRL application to the VRP.

First of all, there are two distinct settings encountered when modelling the system using RL agents. In a single-agent environment, the agent acts as a central dispatcher, simultaneously defining all vehicle routes. In a multi-agent setting, the agent might abstract a driver or be a part of the centralised dispatcher. As a driver abstraction, the agents might need to cooperate to fulfil all the customer's requests or compete to fill the most requests themselves. As a part of the solver,

the route assignment will combine all the individual agent's solutions. Depending on the problem solved by each agent and their methodology, the agents might be either homogeneous (solving the same problem using the same methods) or heterogeneous (solving different parts of the problem or solving the same problem using distinct methods).

Secondly, the authors of [50] differentiate between autoregressive approaches and non-regressive approaches. Both of these approaches present different challenges. Autoregressive techniques construct solutions step-by-step, and each action from the agent corresponds to a step. Usually, this action corresponds to which graph node or customer to visit next or the time to wait at the current position. Non-autoregressive approaches produce the complete solution in the same step. These require extra flexibility since both the input, the environment's state, and the output, the agent's action, do not have a fixed length. Other authors also designate the methods used in these approaches as construction / constructive methods (autoregressive approaches) and improvement methods (non-regressive approaches).

Thirdly, many different algorithms and architectures have been proposed to solve the VRP. Consequently, surveys [28, 93, 129, 162] attempt to examine and catalogue these techniques. Some of the most frequently used methods include REINFORCE, both with the greedy rollout [56, 65, 5, 141] and the rollout baseline [51], actor-critic algorithms, such as A2C [121] and A3C [86], and deep Q-networks, such as DQN [131] and Dueling Double DQN (D3QN) [57].

Furthermore, in an attempt to reduce the number of samples needed by these DRL algorithms, TL has been applied on multiple occasions. In [131], Wang et al. attempt to solve the Pickup and Delivery Problem (PDP) as a central operation task, relying on the DQN framework, using TL to allow learning to be global instead of restricted to a specific city. In this case, they enable knowledge transfer both between cities and across time within the same city. In [5], Ardon explores what they consider to be the two most relevant methods to solve the CVRP. The first includes a Recurrent Neural Network (RNN) encoder-decoder coupled with an attention mechanism from [86], while the second consist of the REINFORCE algorithm from [56]. However, the knowledge transfer experiment only uses the latter. In this experiment, they reuse a pre-trained policy from a more straightforward yet similar problem, the TSP, to solve the CVRP. Similarly, Yaddaden, Harispe, and Vasquez, in [141], also choose the REINFORCE algorithm with a greedy rollout from [56] and attempt knowledge transfer from a pre-trained TSP model to solve the CVRP, varying the instance's size and data generation distribution between the source and target tasks.

Table 3.1 summarises the representation and methodology used in each of the papers. The research studies featured in this table tackle the VRP variants listed in Section 3.1. Additionally, they also address the following TSP variants:

- *Prize Collecting Travelling Salesman Problem* (PCTSP): each city gives a prize when visited and a penalty when it is excluded from the route. The travelling salesman minimises the sum of tour length and penalties while collecting a minimum prize [7].

- *Stochastic Prize Collecting Travelling Salesman Problem* (SPCTSP): a variant of the PCTSP where an expected prize is known, and the real prize is only discovered once we visit the

city [56].

- *Covering Salesman Problem* (CSP): finds a route with a subset of the cities to visit, such that each city outside the route is within a predefined distance of a visited city [15].

The bold lines represent the most relevant papers for this project.

Table 3.1: Summary of Representations and Methodologies used in Deep Reinforcement Learning

| Paper | Routing Problem | State Representation | Action | Reward | Algorithms | Architectures | Uses TL |
|---|---|---|---|---|---|---|---|
| (Vinyals, Fortunato, and Jaitly 2015) [122] | TSP | the customer's coordinates (normalized) | sequence from 1 to N | | pointer network | | |
| (Bello et al. 2017) [10] | TSP | the customer's coordinates (normalized) | | | actor-critic algorithm with policy parametrized by Pointer Network | RNN encoder & decoder | |
| (Dai et al. 2018) [18] | TSP | sequence of actions | new node | change in the cost function | combination of n-steps Q-learning and fitted Q-iteration | graph embedding network: structure2vec | |
| (Nazari et al. 2018) [86] | CVRP, SVRP | | | | REINFORCE, A3C | actor-critic; RNN decoder coupled with attention mechanism | |
| (Wang et al. 2018) [131] | VRPPD | the driver's coordinates and their timestamp | assignment trip-driver | trip fee | central operation task/single driver: DQN framework: Q-learning | | X |
| (Kool, van Hoof, and Welling 2019) [56] | TSP, SPCTSP, CVRP, SDVRP | | | | REINFORCE w/ greedy rollout | attention-based encoder-decoder | |
| (Vera and Abad 2019) [121] | CVRP | customer's demands, their normalized coordinates, and the vehicle's capacity and normalized coordinates | | | A2C | encoder-decoder w/ attention mechanism | |

Table 3.1: Summary of Representations and Methodologies used in Deep Reinforcement Learning

| Paper | Routing Problem | State Representation | Action | Reward | Algorithms | Architectures | Uses TL |
|---|---|---|---|---|---|---|---|
| (Kullman et al. 2019) [57] | VRPSSR | model approach: atari-fication | | | Root Mean Squared Propagation | D3QN | |
| (Zhang et al. 2020) [149] | CVRPSTW | | | | | attention-based encoder-decoder (multi-agent attention model) | |
| (Li et al. 2021) [63] | HCVRP | the customer's coordinates, their demand demands and the vehicle's capacity | | | DRL and attention mechanism | encoder, vehicle selection decoder, node selection decoder (route construction) | |
| (Li et al. 2021) [65] | CSP | the city coordinates | | | REINFORCE w/ greedy rollout | encoder: attention layers (layer = multi-head attention + feedforward) decoder: RNN w/ attention operator | |
| (Zhao et al. 2021) [157] | CVRP, CVRPTW | static (customer's coordinates, service time, time windows) and dynamic (demands, vehicle's load and coordinates) | | | REINFORCE | actor and adaptive critic | |
| (Li et al. 2021) [64] | VRPPD | the partial solution and the customer's coordinates | select next node to visit next | -total travel time | heterogeneous attention mechanism (similar to [56]) | encoder-decoder | |

Table 3.1: Summary of Representations and Methodologies used in Deep Reinforcement Learning

| Paper | Routing Problem | State Representation | Action | Reward | Algorithms | Architectures | Uses TL |
|---|---|---|---|---|---|---|---|
| (Feng et al. 2021) [31] | CVRP, VRPPD | the customer's nodes, their demands, and distance matrix between customers | | | evolutionary multitasking; 1. learning of mapping across CVRPs; 2. knowledge transfer across CVRPs | | |
| (Kim, Park, and Kim 2021) [51] | TCP, PCTSP, CVRP | the partial solution and the customer's coordinates | select unserved task | -tour length | seeder and reviser: REINFORCE w/ rollout baseline | hierarchical problem-solving strategy; 2 iterative DRL policies: seeder and reviser | |
| (Gupta, Ghosh, and Dhara 2022) [42] | single depot multi-customer CVRPTW | | select next customer to visit | | Deep Q-network; NN with experience replay | encoder: graph attention model; decoder: fully connected layers | |
| (Ardon 2022) [5] | CVRP | the graph, the depot node, the customer's nodes and demands, and the vehicle's current node and capacity | select next node to visit | | REINFORCE w/ greedy rollout | attention-based encoder-decoder | X |
| (Yaddaden, Harispe, and Vasquez 2022) [141] | CVRP | the graph, the depot node, the customer's nodes and demands, and the vehicle's current node and capacity | select next node to visit | | REINFORCE w/ greedy rollout | encoder-decoder | |

Table 3.1: Summary of Representations and Methodologies used in Deep Reinforcement Learning

| Paper | Routing Problem | State Representation | Action | Reward | Algorithms | Architectures | Uses TL |
|---|---|---|---|---|---|---|---|
| (Foa et al. 2022) [34] | CVRP | the adjacency matrix, the assignment matrix, the cost approximation, the customer's demand, and the vehicle capacity | select node and assign it to cluster | | PPO | deep convolutional neural network in both actor and critic | X |
| (De Brouchoven de Bergeyck 2022) [20] | CVRP, CVRPTW | the depot and customer's coordinates (normalized) and the customer's demand $\in [0, 1]$ (considering the vehicle capacity = 1) | | | improvement algorithm | encoder | |

In addition, even though there is usually a much bigger focus on methods, algorithms, and architectures, representation still plays a big part in the model's performance. Many studies have tried to understand this impact, and there is a large diversity of methods used. While some methods learn the representation explicitly, others learn it implicitly.

There are a few characteristics that these representations require when applying them to real-world scenarios. These representations need to be able to deal with the same challenges as the RL methods, as showcased in Section 2.1. Table 3.2 enumerates the challenges each of the papers reviewed tackled. It includes a comprehensive list of all the challenges addressed, along with challenges found in [26] and [25]. The first line represents the challenges that the representation would ideally address in order to be applied to the VRP. The bold lines represent the most relevant papers for this project. The names of the columns were shortened to fit the page layout. The first column (*Paper*) includes the name and year of the publication. The last column (*MDP Type*) contains the type of MDP considered by the authors in their work. The other columns represent whether the authors considered the following challenges of RL when applied to the real world.

*LimS* : Limited number of samples for training

*Delay* : Sparse rewards or rewards with unknown or large delays

*HighD* : Highly dimensional state and action spaces

*Constr*: System constraints that should not be violated

*PO* : Partially observable tasks, viewed as non-stationary or stochastic

*MoR* : Multi-objective, unspecified, or poorly specified reward functions

*Fast* : Ability to provide a fast response

*ExtBe* : Training off-line from logs of an external behaviour policy

*Expl* : Explainable policies

*Gener* : Ability to generalise between tasks

*Param* : Avoid millions of model parameters

*Speed* : Agent's learning speed

*IRepr* : Transfer representation between agents with different internal representations

Table 3.2: Summary of Representations in Reinforcement Learning domains and Challenges faced by them

| Paper | LimS | Delay | HighD | Constr | Stoch | MoR | Fast | ExtBe | Expl | Gener | Param | Speed | IRepr | MDP Type |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| important to VRP | X | | X | X | | X | X | X | | X | | | | MDP |
| (Taylor and Stone 2007) [115] | X | | X | | X | | X | X | | X | | X | X | MDP |
| (Stone 2007) [114] | X | | X | | X | | X | X | | X | | X | X | MDP |
| **(Madjiheurem and Toni 2019)** [78] | **X** | **X** | **X** | **X** | **X** | | | | | **X** | | | | **MDP** |
| (Ozair et al. 2019) [90] | X | | X | | | | X | | | | | | | N/A |
| (Rafati 2019) [98] | | X | X | | | | X | | | | | | | MDP |
| (Wang 2020) [123] | | X | X | X | | | X | | | X | | | | MDP |
| (Lu, Huang, and Du 2021) [75] | | X | X | | | | X | | | X | | X | | linear MDP |
| (Yarats et al. 2021) [144] | X | | | | | | X | | | X | | X | | task-agnostic partially observable MDP |
| (Dunion et al. 2022) [27] | | | X | | | | X | | | X | | | | fully-observable MDP |
| (Merckling et al. 2022) [82] | | X | X | | | | X | | | | | | | |
| (Zhang et al. 2022) [147] | | | X | | | | X | | | X | | | | Block MDP |
| (Lan et al. 2022) [60] | X | | X | | | | X | X | | X | | | | MDP |
| **(Joshi et al. 2022)** [50] | **X** | | **X** | | | | **X** | | | **X** | **X** | **X** | | **sequential decision making tasks on graphs** |
| **(Agarwal et al. 2022)** [1] | **X** | **X** | **X** | | | | **X** | | | **X** | | **X** | | **Block MDP** |
| (Alegre, Bazzan, and Silva 2022) [153] | | X | X | | | | X | | | | | | | Block MDP |
| (Zhang et al. 2022) [2] | | | | | | X | X | | | X | | X | | multi-objective MDP |
| **(Zang et al. 2022)** [146] | **X** | | **X** | | | | **X** | **X** | | **X** | | | | **MDP** |
| (Wang et al. 2023) [124] | | X | X | X | | | X | | | X | | X | | MDP |

Finally, we delve into how GDRL has been applied to solving the VRP and other similar problems. As previously discussed, GDRL allows for a variable-sized graph as input, awarding us with extra flexibility and the ability to serve a variable number of customers. Table 3.3 identifies the properties of multiple papers that explore constructive methods. Here, we take a deeper look into the interaction between the agent and the environment. The key aspects of that interaction can be listed as follows:

- *State*: How the environment's state is represented and what node or edge features are used.
- *Action*: What making an action corresponds to. Since we only selected papers using constructive methods, the action always corresponds to selecting the next node. Therefore, this column is omitted from the table.
- *Reward*: What is the reward function, or what it takes into account.
- *Mask*: Whether or not the research paper considers masking invalid actions and how that is done.

The environment's representation will be further discussed in Section 4.1.

Table 3.3: Summary of Representations used in Graph Deep Reinforcement Learning to Solve Routing Problems

| Paper | State | Reward | Mask |
|---|---|---|---|
| (Zhou et al. 2022) [158] | information about the vehicle and the current node and its neighbouring nodes | + award for serving a customer or demand served - total travelled distance - penalty for exceeding the vehicle capacity - penalty for running out of battery - penalty for picking up or delivering a package after the end of the time window | No |
| (Fellek et al. 2023) [30] | node features: 3D vector with the 2D coords of the node and their demand; edges features: euclidean distance between the nodes | | Mask out the depot at step 0; Mask out customers with demand = 0 or demand > remaining vehicle capacity |
| (Stohy et al. 2021) [108] | set of all visited cities | - tour cost | No |
| (Ouyang et al. 2021) [89] | TSP description and sequence of already visited cities | immediate reward: - edge length | No |
| (Kool 2022) [55] | total distance, current node, set of visited nodes (including start node) and remaining capacity of the vehicle | - tour length | Mask out depot when currently in the depot; Mask out customer with demand = 0 or demand > remaining vehicle capacity |
| (Kim, Park, and kim 2021) [52] | partial solution of TSP (sequence of previous actions) | - tour length | No |
| (Yin, Rao, and Zhang 2021) [145] | node embeddings (current node and neighbors) | - edge weight | No |
| (Wang, Hao, and Cao 2021) [126] | set of visited nodes | - tour length | No |
| (Poullet 2020) [97] | static (coordinates) and dynamic (demand) | - total distance | Yes |

| Paper | State | Reward | Mask |
|---|---|---|---|
| (Czuba and Pierzchała 2021) [16] | partial solution and node features | - tour length | No |
| (Kwon et al. 2021) [58] | information about the start and current cities and about all the unvisited cities | | Mask out nodes already visited |
| (Yang et al. 2023) [143] | ordered sequence of visited nodes | - distance cost | Mask visited city nodes |
| (Zhang et al. 2021) [155] | partial solution of subproblem, candidate nodes and position, current capacity, total distance, and total travel time of each vehicle | 2 conflicting objectives: the total traveling distance and makespan | Mask out nodes that would violate constraints or were already served |
| (Singh et al. 2023) [106] | local traffic conditions | | No |
| (Sykora, Ren, and Urtasun 2020) [113] | | - total distance | Mask out nodes already visited |
| (Boffa et al. 2022) [11] | nodes: customers locations and demands; edges: existence and distance of connection between customers | - tour length (distance) | Mask out nodes already visited |
| (Lin, Yang, and Zhang 2022) [69] | nodes embeddings for all cities | multiobjective | Mask out nodes already visited |
| (Wu et al. 2021) [138] | depot and customer coordinates, customer demands, drone capacity and maximum flight range | - route distance | Mask out depot when currently in the depot; Mask out customer with demand = 0 or demand > remaining vehicle capacity; Mask out customers outside range |
| (Strauss et al. 2023) [109] | resources: availability and position; agents: ID and position; env: holidays and time of the day | number of resources collected by the agent (assumes that travelling between resources takes time) | No |

| Paper | State | Reward | Mask |
|---|---|---|---|
| (Chen et al. 2023) [13] | current partial solution, unvisited customers, and remaining capacity | - route distance | Mask out customers already visited or violating capacity constraints |
| (Sultana et al. 2022) [110] | sequence of visited nodes | - tour length | Mask out nodes already visited |
| (Oren et al. 2021) [88] | partial solution | - edge weight | Mask out invalid nodes (allows noop actions) |
| (Lei et al. 2022) [62] | partial tour and vehicle inventory | change in tour cost ( - edge weight ) | Mask out depot when currently in the depot; Mask out customer with demand = 0 or demand > remaining vehicle capacity |
| (Wang, Lai, and Tang 2023) [128] | set of all possible actions / nodes and their embeddings | - tour length | Yes |
| (Zhang et al. 2023) [156] | static (traffic patterns and customer IDs, coordinates and demands); dynamic (current customer pool, customer visited status, and current traffic conditions) | - edge travel time | Mask out customers already visited, violating capacity constraints or whose order hasn't been picked up |
| (Zhang, Wang, and Zhang 2022) [154] | partial solution (sequence of visited nodes) | - edge distance | Yes, with capacity and time window constraints |
| (Hottung, Kwon, and Tierney 2022) [46] | starting state: positions of the cities and starting city; partial: sequence of actions | - tour length (at the end) | No |
| (Zhang, Lin, and Li 2023) [151] | agent & environment states | - travel cost | Mask out customers already served or whose capacity exceeds the vehicle remaining capacity; Mask out other depots (vehicles can only return to the starting depot) |
| (Wu et al. 2023) [140] | static: graph embedding; dynamic: context embedding and previously select flights | - tour length (at the end) | Mask out nodes already served, whose demands exceeds the current vehicle capacity or whose service cannot be finished beore the end of the time window; Mask out depot when in the depot if there are nodes that can be visited |

| Paper | State | Reward | Mask |
|---|---|---|---|
| (Salgo, Banks, and Rivest 2021) [102] | nodes: pairs of weapon-targets; node features: weapon features, target features, and how they related to one another | | No |
| (Hou et al. 2023) [47] | current partial solution | - travel distance | Mask out the depot when the remaining vehicles cannot serve all the demand; Mask out the customers that cannot be served by the current vehicle; Mask out nodes when the vehicle cannot arrive at the customer inside its time window |
| (Wu et al. 2021) [139] | static (customer information, time windows and traffic patterns); dynamic (current traffic conditions and visited status of customers) | - total distance - penalty * number TW violations - huge penalty * is feasible solution | No |
| (Lee and Jang 2022) [61] | | reward shaping: uses the immediate reward (edge travel time) and a potential function (the deterministic shortest time to traverse the edge) | No |
| (Hendriks 2022) [45] | coordinates and visited nodes in a flipped fashion (using the last one in the beginning of the fixed sized vector, filling it with 0 in the right) | - edge distance | No |
| (Wu et al. 2022) [136] | set of visited zones | maximize the model's conditional probabilities as per the learned Prediction by Partial Matching model | No |
| (Alharbi et al. 2022) [3] | partial solution | - tour distance | Mask out customers already served, or whose demand exceeds the remaining capacity; Mask out pickup cities if the adding the demand and remaining capacity exceeds the vehicle maximum capacity; Mask out the depot when in the depot |

| Paper | State | Reward | Mask |
|---|---|---|---|
| (Peng, Wang, and Zhang 2020) [95] | partial solution of instance and node features | - tour length | Mask out the depot when in the depot; Mask out customers with demand = 0 or demand > remaining vehicle capacity |
| (Ling, Zhang, and Chen 2023) [70] | image representation | - edge Euclidean distance | No |
| (Luo et al. 2022) [76] | generated subtour | - tour length | Mask out nodes already visited |
| (Fu et al. 2022) [37] | node coordinates, time windows, service time and emergency state (capturing demand and emergency state together) | - tour length - penalties for number of customers served outside their time window | Mask out nodes already visited |
| (Zhang et al. 2023) [148] | customer node: coordinates and demand; route construction: remaining capacity; current partial solution: current node | - tour length | Mask actions that lead to infeasible solutions |
| (Zhang et al. 2023) [150] | unvisited customers and current agent state (location and remaining capacity) | - total travel cost | Yes, to ensure that the pick up is served before the paired delivery customer |
| (Wang 2021) [125] | set of visited nodes | - tour length | No |
| (Zhu et al. 2023) [159] | set of visited nodes | | No |
| (Kool, Van Hoof, and Welling 2019) [56] | node locations | difference between the costs of the selected solution and the baseline | Mask out depot when currently in the depot; Mask out customer with demand = 0 or demand > remaining vehicle capacity |
| (Ma et al. 2019) [77] | set of all visited cities | - edge distance | No |
| (Cao, Sun, and Sartoretti 2022) [12] | city and agents coordinates relative to observing agent and a global mask | - max tour length (among agents) | Yes |

| Paper | State | Reward | Mask |
|---|---|---|---|
| (Lin, Ghaddar, and Nathwani 2022) [67] | node information: coordinates, time windows and demand; global information: time and battery of the vehicle, and number of vehicles available | - total distance - penalties for constraint violation - penalties for charging station visits | Mask out the depot when in the depot (except if there is no remaining demand); Customer has demand = 0 or demand > vehicle remaining capacity; Remaining battery cannot support travelling to the customer and then to the depot; Earliest arrival violates the time window constraint; If the vehicle travels to the node, it will not be back at the depot before the end of the planning horizon |
| (Liu et al. 2022) [72] | location, remaining energy and unvisited points | - edge energy cost | Yes |
| (Liu, Shin, and Tsourdos 2023) [73] | node features: location; edge features: connectivity and distance | - energy costs | Yes |
| (Wang, He, and Tang 2023) [127] | ordered sequence of visited nodes | K - edge weight | No |

## 3.3 Discussion

Although the existing literature on solving the VRP using DRL is extensive, its main focus is on different algorithms and architectures. Therefore, there are many studies exploring the performance correlation of different algorithms. Moreover, multiple studies have demonstrated the impact of leveraging knowledge from one or more source domains to improve their model's performance on the target domain. When transferring knowledge, the authors usually share the policy or value function and compare the performance with and without using TL. However, there is still a lot of uncertainty about precisely what causes TL to have negative effects.

Even though representation plays a vital role in both RL and TL, there is much less research on the impact of representation on the model's performance. From observing Table 3.2, only one of the featured papers focused on the TSP and none on the VRP domain. Besides, only one study [2] considered a multi-objective reward function, and none explored all the concepts we deemed ideal to fit a VRP variant.

Graph Deep Reinforcement Learning grants large input flexibility, allowing the use of variable-sized graphs seamlessly. Since the VRP can be represented in a graph form, creating an environment to interact with a GDRL agent is natural. Nonetheless, there is still less research on GDRL than DRL or other methods due to it being relatively recent. Additionally, only a few of the papers researched contained GDRL and TL, even though it was considered a viable way to improve generalisation [142, 71]. Besides, there is a lack of understanding regarding the effects of the reward function on the learning and exploration phase, and finding a good reward function to guide the RL agent is still an open area of research.

## 3.4 VRP Taxonomy

From all the papers reviewed, we established a taxonomy for solving the VRP using RL. This taxonomy is divided into four parts:

- Components
- VRP
- VRP + RL
- RL

However, it is worth noting that the RL taxonomy was already covered in Chapter 2. Therefore, we will refrain from discussing it again.

Starting with the components, the VRP has three main inter-connected components or phases that can be solved sequentially or simultaneously: assignment, bin-packing, and routing. A simplified view of the components, their classifications, and examples can be seen in Figure 3.1.

The assignment subproblem defines pairs of (package, vehicle) for all packages. We can consider this assignment to be either a single package per vehicle (one-to-one or balanced) or multiple packages per vehicle (many-to-one or unbalanced). Ideally, the assignment would take into account the vehicle's maximum capacity and the delivery locations in order to favour the subsequent phases.

The bin-packing subproblem is responsible for fitting all the assigned packages inside the vehicle in the most convenient order. For the bin-packing, multiple dimensions can be considered, depending on the packages in question and their transportation requirements.

- 1D considers only the package weight.
- 2D disregards the package weight but considers that packages cannot be placed on top of one another.
- 3D either disregards the weight or considers that packages cannot be placed on top of one another.
- 4D considers the package's weight and 3D size, considering that they might be placed on top of each other.

Additionally, the most convenient order might consider the delivering order, package properties (such as the ability to turn packages, placing heaviest ones on the bottom, or placing lighter or fragile ones at the top), or both.

The routing subproblem aims to minimise the cost of the trip while serving all the customers. As already mentioned in Section 3.1, the cost might vary. It might represent the total distance travelled, total time spent, battery or energy used, fuel consumed, $CO_2$ or GHG emissions, total customer waiting time, maximum customer waiting time, or even a combination of multiple objectives. The routing component is greatly affected by the graph and its underlying structure and characteristics, namely size, topology, and node and edge properties. Besides, it is also affected by connectivity restrictions or limitations, such as traffic regulations. These connectivity limitations include the following:

- Fixed or permanent. For example, cars can never pass through a pedestrian pathway.
- Time-based. For example, vehicles may be unable to pass through a specific area at night.
- Temporary disruptions:
  - Planned. For example, due to construction, processions, fairs, or other sporting or seasonal events.
  - Unplanned. For example, due to incidents, protests, roadblocks, or accidents.

Additionally, these connectivity limitations might affect the entirety of the fleet or just some of its vehicles (when the fleet is heterogeneous).

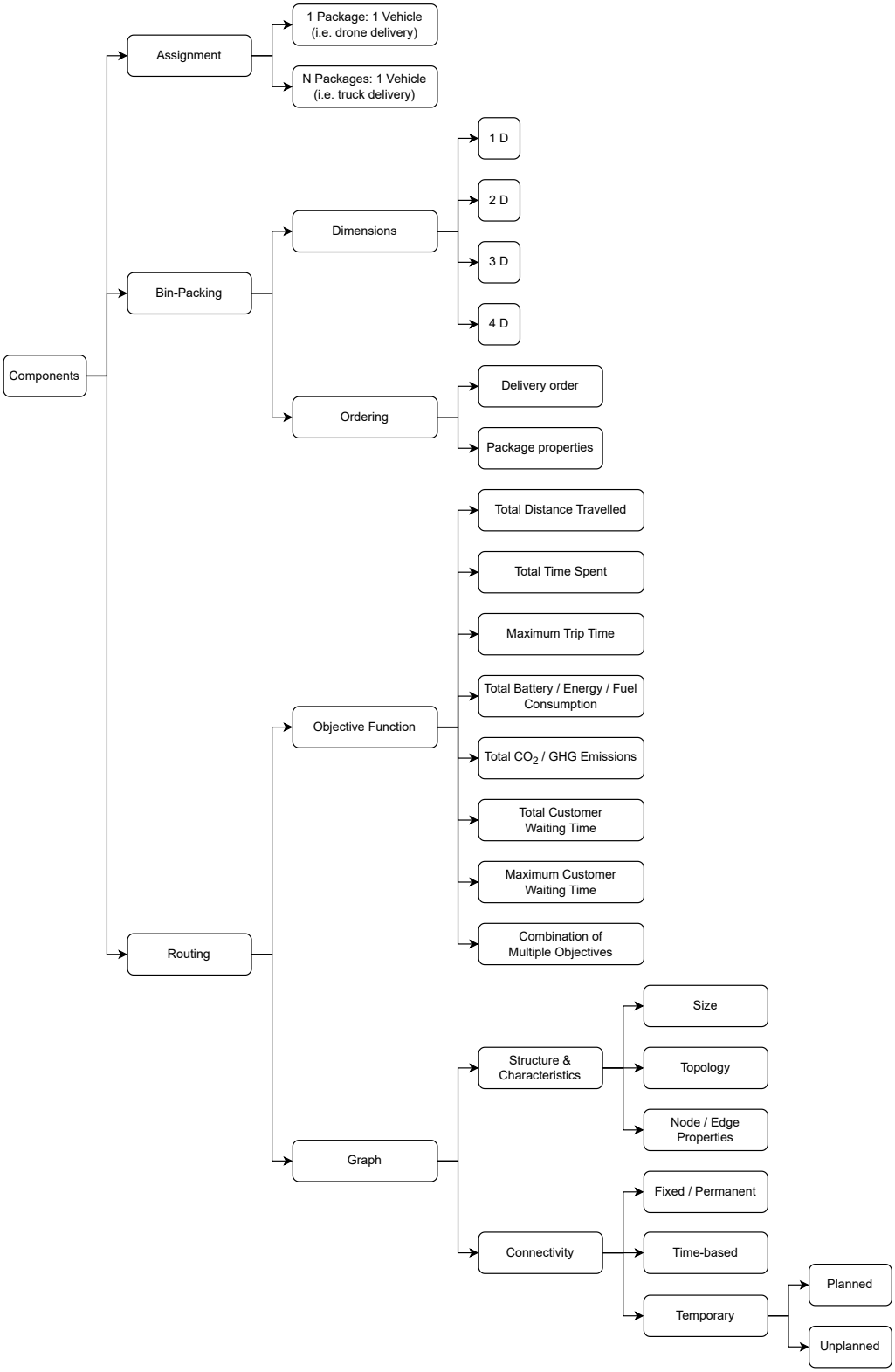


Figure 3.1: Vehicle Routing Problem Components

Moving on to the VRP portion, we complement the variants listed in Section 3.1. We enhance the already listed variants with a constraint superset and a classification of their operation mechanisms. Figure 3.2 portrays an overview of the divisions of the VRP part.

Each VRP variant can be considered as a VRP with a set of extra constraints from the following superset:

- *Maximum vehicle capacity*: limiting the amount of cargo each vehicle can carry, either in weight, volume, or both.

- *Customer time windows*: requesting the delivery of packages during a specified time window, either obligatory or optional (at the cost of penalties).

- *Pickup and delivery*:

  - *Pickup and delivery of the same package*: the package must be picked up from the pickup point before being delivered to the delivery point.

  - *Pickup and delivery of different packages*: the vehicle might pick up a package and deliver a different one during the same trip. It can be considered a specific case of the *pickup and delivery of the same package* when the package to be picked up is delivered at the depot, and the package to be delivered is picked up at the depot.

- *Customers served*: serving all the customers might be obligatory or optional (at the cost of penalties).

- *Stochastic customers*: the customers might have stochastic demand, location, or both.

- *Limited vehicle battery / energy*: the vehicle might have a limit on the distance it can travel before returning to the depot or visiting a recharging station.

Additionally, the VRP can also be distinguished based on its operating mechanisms. Firstly, we consider the fleet and its characteristics, the number of vehicles (either finite or infinite), the number of trips per vehicle (single trip or multiple trips), the homogeneity (either homogeneous or heterogeneous), the type of vehicles (combustion, electric, hybrid or drones), and its crew (either crewed or crewless). Secondly, we consider the delivery types. Each delivery point might allow either a single package or multiple packages (split delivery). Finally, considering the number of depot locations, the VRP might have a single or multiple depots.

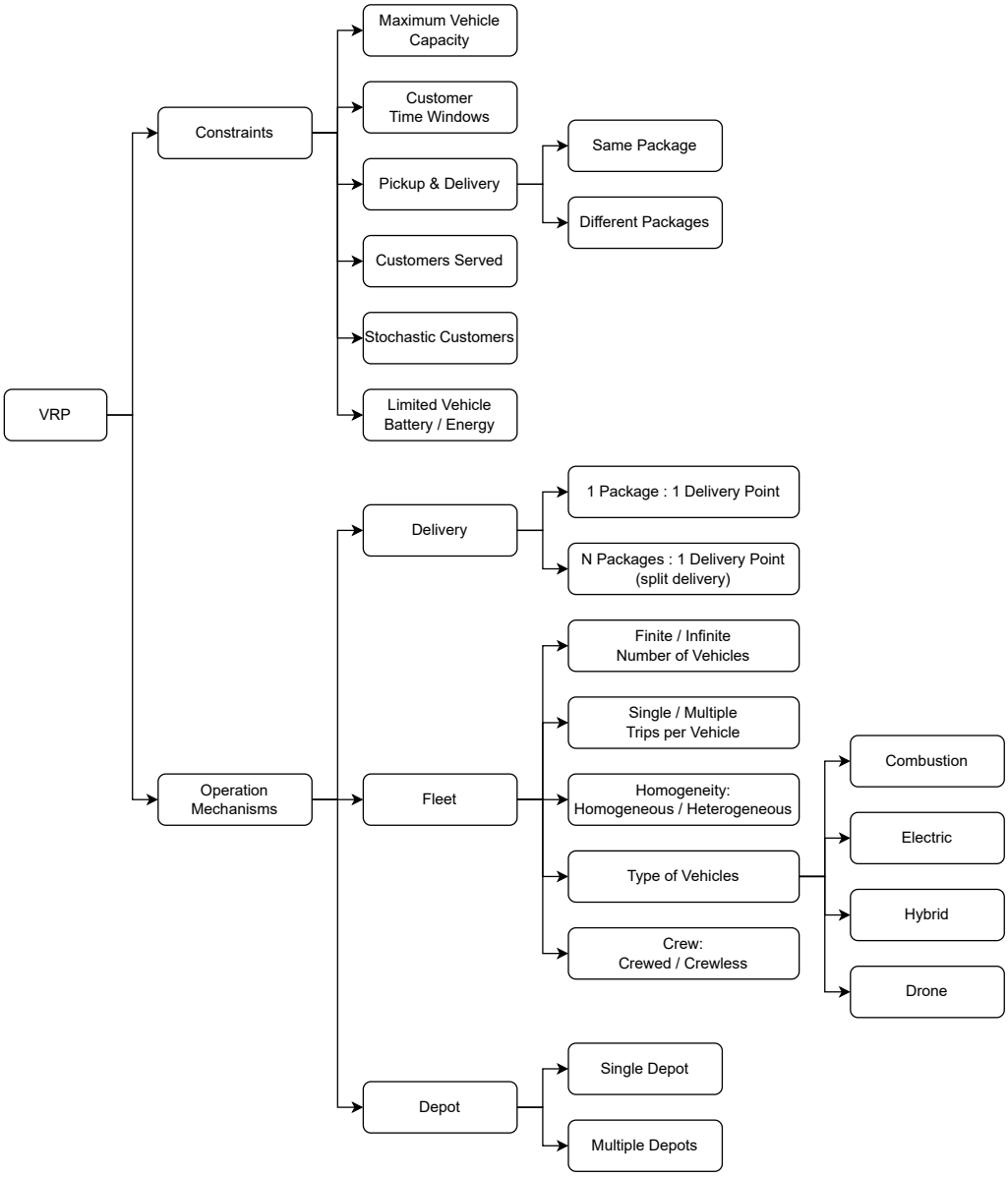


Figure 3.2: Constraints and Operation Mechanisms Used when Addressing the Vehicle Routing Problem

Last but not least, we characterise the VRP + RL portion. This part incorporates the necessary assumptions required to solve the VRP using RL. An overview is provided in Figure 3.3.

Regarding decision-making, we can divide the methods into centralised and decentralised. Regarding representation, they can be grouped based on the state, action, and reward. The state might have a fixed or variable size. When the state is represented as a graph, it might have a variable size. The action might select the next node or customer, making the method a constructive method, or the complete route, making the method an improvement method. The reward might be single objective or multiobjective and might in consideration classic, traditional or economic optimisation metrics (total distance travelled or total time spent), ecological metrics (battery /

energy / fuel consumption, or $CO_2$ / GHG emissions), or customer satisfaction (total customer waiting time or maximum customer waiting time).

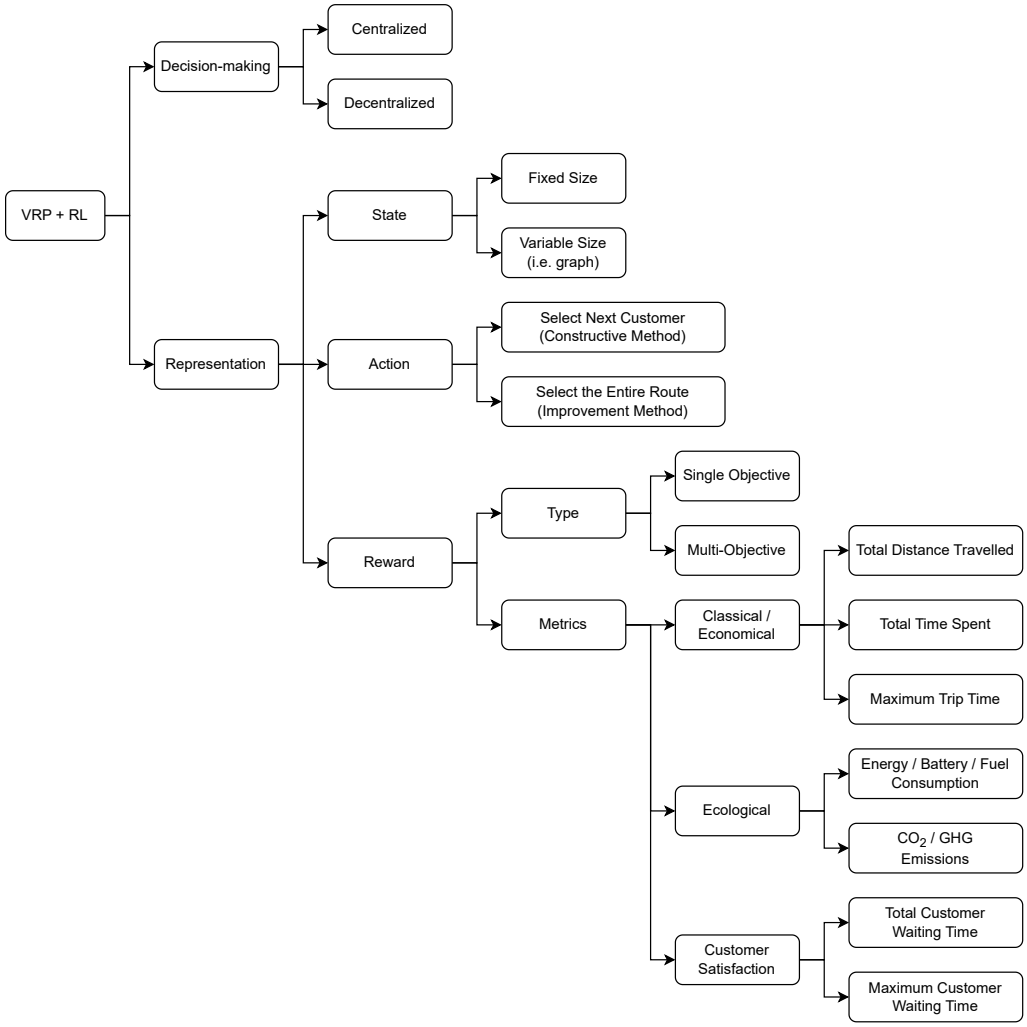


Figure 3.3: Assumptions Required to Solve the Vehicle Routing Problem using Reinforcement Learning

# Chapter 4

# Methodological Approach

This chapter previews how to tackle the problem. It starts with a formalisation of the problem constraints. Next, we dive into how to represent the environment in a graph form and highlight useful node features. Afterwards, we describe the implementation of a masking mechanism to narrow down the action space. Later, we explore different reward functions and how reward shaping affects the learning phase. Lastly, we examine evaluation metrics which can be employed to measure the agent's performance, the efficiency of its actions and overall decision-making capabilities.

## 4.1 The Environment

The research problem addressed can be framed as an RL problem. In RL, an agent aims to optimise the reward it receives from the environment over the sequence of actions it executes. Essentially, it learns how to convert the state's representation into the most profitable or desirable action. As such, an RL agent can be formalised as an MDP, explained in Section 2.1.

Hence, the solution to our research problem can be seen as the sequence of actions during an episode. Notwithstanding, we want the solution to respect all the constraints applied to the VRP variant in question. In this project, we experiment with two variants: the Capacitated Vehicle Routing Problem (CVRP) and the Capacitated Vehicle Routing Problem with Soft Time Windows (CVRPSTW).

Consider,

- $G = (V, E)$: the VRP variant graph.
- $V = 0, \ldots, n$: the graph vertices, where 0 is the depot and $1, \ldots, n$ are the customers.
- $E = (i, j) : i, j \in V$: the graph edges. In edge $(i, j)$, $i$ represents the origin vertex, and $j$ is the destination.
- $d_{i,j}$: the distance travelled when traversing edge $(i, j)$.

- $tt_{i,j}$: the travel time when traversing edge $(i, j)$.
- $c_i$: capacity required / demand at vertex $i$.
- $s_i$: the start of the time window for delivery at vertex $i$.
- $e_i$: the end of the time window for delivery at vertex $i$.
- $st_i$: the service time (time taken to deliver the demand) at vertex $i$.
- $C_k$: the maximum capacity of vehicle $k$.
- $K$: the vehicle fleet.
- $X_k$: the number of time steps used by vehicle $k$.
- $trip_x^k$: the vertex visited by vehicle $k$ at time step $x$. Consequently, $trip_0^k$ and $trip_{X_k}^k$ are the first and last locations visited by the vehicle, respectively.

The basic VRP constraints pertain to routing. These ensure that the trips are valid (they start and end at the depot) and all the customers are served (all the delivery locations are visited). These constraints are present in any VRP variant and can be formulated as follows:

- Each vehicle leaves and returns to the depot.

$$\forall k \in K, trip_0^k = 0 \wedge trip_{X_k}^k = 0 \tag{4.1}$$

- Each delivery location is visited exactly once.

$$\forall i \in V \setminus \{0\}, \sum_{k \in K} \sum_{x=0}^{X_k} \left( \begin{cases} 1 & \text{, if } trip_x^k = i \\ 0 & \text{, otherwise} \end{cases} \right) = 1 \tag{4.2}$$

The CVRP includes the basic VRP restrictions and an additional one related to the vehicle's capacities. This capacity constraint considers that the vehicle has limited space or a weight limit (all the packages carried by the vehicle cannot surpass the maximum vehicle capacity). In short, we consider the following constraints:

- Each trip must start and end at the depot.
- Each delivery location must be served exactly once.
- The capacity of each vehicle cannot be exceeded.

This capacity constraint can be formulated as follows:

$$\forall k \in K, C_k \geq \sum_{i \in V} \left[ \sum_{x=0}^{X_k} \left( \begin{cases} 1 & \text{, if } trip_x^k = i \\ 0 & \text{, otherwise} \end{cases} \right) * c_i \right] \tag{4.3}$$

In the CVRPSTW setting, in addition to the previously presented constraints, there is another one related to the customer's requested time window. This restriction requires the package delivery to be done within the requested time window. However, in opposition to the previous restrictions, which are hard constraints since they cannot be infringed, this time window restriction can be violated at the cost of collecting a penalty, making it a soft constraint. In short, we consider the following constraints:

- Each trip must start and end at the depot.
- Each delivery location must be served exactly once.
- The capacity of each vehicle cannot be exceeded.
- The packages should be delivered within the customer's requested time window. Failure to comply will result in a penalty.

The last restriction can be formulated as follows:

$$\forall k \in K, \forall x \in \{0..X_k\}, s_{trip_x^k} \leq \sum_{y=0}^{x-1} \left( st_{trip_x^k} + tt_{trip_x^k} \right) \leq e_{trip_x^k} \tag{4.4}$$

### 4.1.1 Node Features

We represent the environment in a graph form. In this graph, the nodes correspond to the points of interest, the depot and the delivery locations, and the edges correspond to the connection between each pair of points. We can characterise each node with a group of attributes named node features and each edge with a weight (the distance between the points).

We divide the node features into two categories. The first category covers the attributes which can be used to characterise nodes in both settings (CVRP and CVRPSTW). The attributes which can only characterise nodes in the second scenario (CVRPSTW) are incorporated into the second category.

The first class of attributes includes:

- *location*: whether or not the vehicle is at the node.
- *depot*: whether or not the node is the depot.
- *demand*: the current demand of the node (0 at the depot or when the customer was already served).
- *current_capacity*: the vehicle's current capacity.
- *next_capacity*: the maximum capacity of the next trip.
- *possible_capacity*: the vehicle's capacity after travelling to the node and serving the node's customer.

- *can_serve*: whether or not the vehicle has enough capacity to serve the customer at the node.
- *fuel_consumption*: the fuel consumption when travelling to the node (from the current location).
- *cost*: the distance cost of travelling to the node (from the current location).
- *mask*: whether or not this node would be masked out.

The second class of attributes includes:

- *current_time*: current time.
- *travel_time*: time taken to travel to the node (from the current location).
- *service_time*: the time taken to serve the node (or reload the vehicle at the depot).
- *start_tw*: the start time of the node's time window.
- *end_tw*: the end time of the node's time window.
- *arrival_time*: the time of arrival at the node (when travelling to the node directly from the current location).
- *time_after_service*: time after serving the customer at the node (when travelling to the node directly from the current location).
- *inside_tw*: whether or not the vehicle arrives at the node inside its time window (when travelling to the node directly from the current location).
- *tw_error*: the number of seconds the vehicle misses the time window for (when travelling to the node directly from the current location). This value is negative when the vehicle arrives before the time window's start and positive if the vehicle arrives after the time window's end.
- *after_tw_start*: whether or not the vehicle arrives at the node after the start of its time window (when travelling to the node directly from the current location).
- *before_tw_end*: whether or not the vehicle arrives at the node before the end of its time window (when travelling to the node directly from the current location).

### 4.1.2 Masking Scheme

The action of our RL agent is equivalent to picking the next point of interest (depot or customer) to which the vehicle will travel. Ideally, we would want this selected point to lead to the optimal solution. However, first, the agent needs to learn. With only the reward function as a guide, the agent is forced to explore a very large action space. Besides, not all the nodes in the graph lead to a

feasible solution at all time steps. Since the research problem we are studying comprises multiple hard constraints, these could be incorporated into the agent's decision-making process.

In fact, in the literature, there have been many occasions where masks have been utilised to block out invalid actions from being selected by the agent. The authors of [48] demonstrate the positive effects of masking by comparing the performance of *invalid action masking* (removing the actions from being selected) with *invalid action penalty* (giving a negative reward for invalid actions), especially in large invalid action spaces. In Section 3.2, we previewed some of the masks utilised in the literature to mask out actions in routing problems. These masking schemes range from blocking out the possibility of staying in the same location forever to blocking out all the points that turn the solution infeasible or that are known to lead to a non-optimal solution. For example, there might be a time when the agent decides to explore whether or not it should leave the depot or a particular customer. However, we are already aware of the outcome of this (staying in the same location for multiple steps in a row). It yields unsatisfied customers and unfeasible solutions.

Therefore, we implement a masking mechanism to limit the nodes that the agent can select. This mask effectively narrows down the number of actions the agent can select. As a result, the agent can only select nodes that would lead to a valid solution. Indeed, the mask blocks the agents from executing the following actions:

- Not moving (selecting the current node as the next node).
- Visiting a customer that has already been served (selecting a node that is not the depot and has no demand).
- Visiting a customer that it cannot serve (selecting a node whose demand exceeds the vehicle's remaining capacity).

Which leads our agent to adopt the behaviours:

- At each time step, it travels to a node different from the one it is currently in.
- At each time step, it either serves a customer or reloads at the depot.

In short, we mask out actions $a$ whose nodes are considered invalid by the following equation.

$$\text{Invalid}_a : (a = \text{current\_location}) \vee (a \neq \text{depot\_node} \wedge \neg(0 < \text{demand}_a \leq \text{current\_capacity})) \quad (4.5)$$

where,

- $Invalid_a$ determines whether the action is invalid or not.
- $current\_location$ is the node the vehicle is currently in.
- $depot\_node$ represents the depot node (usually 0).
- $demand_a$ corresponds to the demand of the node selected by action $a$.

- *current_capacity* is the remaining vehicle capacity.

Nevertheless, this mask is only defined to limit the actions based on hard constraints. In the CVRPSTW, the restrictions related to the customer's requested time windows are soft constraints. Since soft constraints can be infringed, selecting an action that violates them does not induce an unfeasible solution. Therefore, they were not masked out. While they lead to penalties, breaching them might be necessary to fulfil the demand. Besides, the agent is required to make decisions about whether or not it is beneficial to break these restrictions.

### 4.1.3 Reward Functions

As previously stated, reward functions help the agent learn the optimal policy by guiding it towards the goal. Sparse rewards, even though the most direct and general way to design the reward for many problems, might prove challenging to train with [92]. When the RL agent uses a sparse reward function, some issues that might emerge include:

- The agent does not know which action or subsequence of actions better influenced the reward it obtained. This problem is also known as the credit assignment problem or the blame attribution problem.
- The agent might be unable to find any reward in the environment. Without any reward, the agent cannot learn.
- The agent might be unable to find any positive reward in the environment. Without positive rewards, the agent will favour inactivity or inexistent rewards (actions with a reward value of 0).

A better-shaped reward function that distributes the rewards over multiple actions without modifying the intended optimal policy can speed up training and help the agent learn the optimal policy more efficiently.

Reward shaping is a technique used to enhance the natural and direct reward design. It adds intermediate rewards to compliment the agent when it progresses towards the goal or penalise it when it moves further away. This new reward function is much denser and can help the agent try out promising behaviours earlier. However, it risks distracting the agent from the real goal and learning a sub-optimal policy, even when the shaping is very close to the actual value [134].

When solving a VRP, reward functions are used to communicate to the agent the quality of the solution it found. Higher rewards indicate a better solution. In this project, the reward functions employed are a linear combination of reward signals. Therefore, we establish three groups of reward signals.

The first group encourages the agent to serve customers. It consists of a single function called *Step Reward*. It awards the agent for serving a customer while penalising it for returning to the depot and travelling to customers that it cannot serve (either by lack of capacity or because the

customer was already served). The purpose of this reward signal is twofold. Firstly, it acts as a safeguard, incentivising the vehicle to avoid travelling to the depot after serving a single customer. Secondly, it guides the agent away from customers it has already served (when learning without the mask). The *Step Reward* can be formalised as follows:

$$\text{Step Reward}_a = \begin{cases} -1 & \text{, if destination}_a = \text{depot} \\ 1 & \text{, if } 0 < \text{demand}_{destination_a} \leq \text{current\_capacity} \\ -2, & \text{otherwise} \end{cases} \tag{4.6}$$

, where

- $a$ is the action that the agent executed.
- $destination_a$ is the destination node after executing action $a$.
- $depot$ is the depot node.
- $demand_{destination_a}$ is the customer demand in the node selected by action $a$.
- $current\_capacity$ is the remaining vehicle capacity before executing action $a$.

The second group evaluates the routing quality, focusing on metrics related to the distance travelled and fuel consumption. It comprises five different reward signals, *Distance Reward*, *Normalised Distance Reward*, *Distance Fraction Reward*, *Distance Savings Reward*, and *Fuel Consumption Reward*. The first four rewards account for the total distance travelled, an economic metric, while the fifth accounts for fuel consumption, an ecological metric.

The first three penalise the agent based on the travelled distance between the current location ($current\_location$) and the destination node ($destination_a$) selected by action $a$. When the vehicle travels longer, all of them will provide the agent with a smaller reward. The *Distance Reward* penalises the agent proportionally to the distance travelled, as detailed in Equation 4.7.

$$\text{Distance Reward}_a = -\text{d}_{current\_location, destination_a} \tag{4.7}$$

, where $d_{current\_location, destination_a}$ is the distance travelled by the vehicle to go from $current\_location$ to $destination_a$.

The *Normalised Distance Reward* normalises the distances in order to obtain reward values between 0 and 1, following the formula:

$$1 - \frac{\text{d}_{i,j} - \min_{i,j}(\text{d}_{i,j})}{\max_{i,j}(\text{d}_{i,j}) - \min_{i,j}(\text{d}_{i,j})} \tag{4.8}$$

, where $\min_{i,j}(\text{d}_{i,j})$ represents the minimum distance between any two points and $\max_{i,j}(\text{d}_{i,j})$ represents the maximum distance between any two points. Since the distances cannot be negative, and the distance between any location and itself is always 0, the minimum value will always be 0. Hence, Equation 4.9 defines the *Normalised Distance Reward*.

$$\text{Normalised Distance Reward}_a = 1 - \frac{\mathrm{d}_{i,j}}{\max_{i,j}(\mathrm{d}_{i,j})} \tag{4.9}$$

The *Distance Fraction Reward* is a customised reward function based on the distance travelled by the vehicle and the properties of fractions. This reward is also characterised by having its values in the $[0,1]$ range. It is defined in Equation 4.13. The mathematical explanation for this formula is presented below in Equations 4.10, 4.11, and 4.12, where $i$ and $j$ represent any two points. The value 1 added between Equations 4.10 and 4.11 allows us to avoid divisions by 0.

$$0 \leq \mathrm{d}_{i,j} < \infty \tag{4.10}$$

$$1 \leq \mathrm{d}_{i,j} + 1 < \infty \tag{4.11}$$

$$0 < \frac{1}{\mathrm{d}_{i,j} + 1} \leq 1 \tag{4.12}$$

$$\text{Fraction Reward}_a = \frac{1}{\mathrm{d}_{i,j} + 1} \tag{4.13}$$

The *Distance Savings Reward* is another custom reward function inspired by the Clarke-Wright Savings algorithm [14]. The Clarke-Wright Savings algorithm starts with the trivial solution of making a trip per customer. Then, iteratively, selects an edge to connect two different routes. This edge is determined based on which edge would incur a bigger decrease in the overall travel distance. Similarly, the *Distance Savings Reward* awards and penalises the agent based on the reduction in the travel distance. It starts by assuming the same scenario: the vehicle travels to each delivery point and back to the depot. In this scenario, we do not consider multiple trips to the same location, as the vehicle only needs to deliver a single package by location. As such, subsequent visits are not serving the customer and are, therefore, useless. As the agent selects new nodes to add to its partial route, it calculates the savings between the actual path it follows and the trivial one, similar to Clarke-Wright Savings. Firstly, it starts by calculating the cost of travelling to the node selected by action $a$ using the trivial worst-case scenario:

$$\text{expected\_cost}_{current\_location,destination_a} = \mathrm{d}_{current\_location,depot} + \mathrm{d}_{depot,destination_a} \tag{4.14}$$

Then, it calculates the actual cost of travelling to the node by selecting action $a$. Note that if the customer is not served, the vehicle will need to come back, leading to an increase in the total distance travelled. The cost of this extra trip (travelling to the depot and back to the customer) is calculated in the same way as the expected cost.

$$\text{real\_cost}_{current\_location,destination_a} = \mathrm{d}_{current\_location,destination_a} + \text{non\_served\_cost}_a \tag{4.15}$$

$$\text{non\_served\_cost}_{destination_a} = \begin{cases} 0 \text{ , if } 0 < \text{demand}_a \leq \text{current\_capacity} \\ \text{d}_{destination_a,depot} + \text{d}_{depot,destination_a} \text{ , otherwise} \end{cases} \tag{4.16}$$

Finally, Equation 4.17 defines the Distance Savings Reward.

$$\begin{aligned} \text{Distance Savings Reward}_a = \text{expected\_cost}_{current\_location,destination_a} \\ - \text{real\_cost}_{current\_location,destination_a} \end{aligned} \tag{4.17}$$

The last reward of this group is the *Fuel Consumption Reward* depicted in Equation 4.23. As further explored in Section 4.2.3 and Equation 4.30, the fuel consumption can be calculated using the formula:

$$\text{Fuel Consumption} = \text{Distance Travelled} * \text{Vehicle and Cargo Weight} \tag{4.18}$$

Accordingly, the *Fuel Consumption Reward* attempts to reduce the magnitude of the values obtained using the maximum possible fuel consumption for a single trip segment between any two nodes. This worst possible value (*max_fuel_partial*) corresponds to travelling the graph edge with the biggest length while the vehicle is full.

$$\text{max\_fuel\_partial} = (\text{vehicle\_weight} + \text{vehicle\_capacity}) * \max_{i,j}(\text{d}_{i,j}) \tag{4.19}$$

, where:

- *vehicle_weight* corresponds to the weight of the vehicle without any packages. In this project, we assumed a vehicle weight of 1500 kilograms. While this is not close to the weight of a truck, it is a good approximation for the poundage of a last-mile delivery vehicle with smaller sizes. Besides, on average, in the Amazon dataset (described in Section 5.1), the vehicles have a maximum capacity of 3.6 cubic metres.

- *vehicle_capacity* corresponds to the maximum capacity of the vehicle.

For each action selected, the vehicle travels from its current location to the location chosen by action $a$. However, the packages delivered at the destination need to be carried from the depot to that location. Therefore, when the agent adds a new customer to the trip, the corresponding package needs to be transported from the depot until it gets dropped off.

$$\text{fuel\_consumed}_{vehicle} = \text{d}_{current\_location,destination_a} * \text{vehicle\_weight} \tag{4.20}$$

$$\text{fuel\_consumed}_{package} = \text{distance\_since\_depot}_{destination_a} * \text{package\_weight} \tag{4.21}$$

, where $distance\_since\_depot_{destination_a}$ corresponds to the route's total distance from the depot to the node selected by $a$, passing by all the other nodes previously selected after the last depot visit.

$$\text{fuel\_consumed}_{current\_location,destination_a} = \text{fuel\_consumed}_{vehicle} + \text{fuel\_consumed}_{package} \tag{4.22}$$

Finally, Equation 4.23 depicts the *Fuel Consumption Reward*.

$$\text{Fuel Consumption Reward}_a = 1 - \frac{\text{fuel\_consumed}_{current\_location,destination_a}}{\text{max\_fuel\_partial}} \tag{4.23}$$

Instead of using the maximum distance of a single segment and considering the vehicle full during that segment, one could use other values to reduce the magnitude of the reward value. Another possibility includes considering the maximum fuel consumed in a trip to deliver a single package of capacity equal to the vehicle's maximum capacity, as shown below.

$$\begin{aligned}\text{max\_fuel\_partial}_{alternative} = \max_i(&\text{d}_{depot,i} * (\text{vehicle\_weight} + \text{vehicle\_capacity}) \\ &+ \text{d}_{depot,i} * \text{vehicle\_weight})\end{aligned} \tag{4.24}$$

The third group penalises the agent for missing its time windows and measures customer satisfaction. It includes two different reward signals, *Error Reward* and *Inside or Outside Reward*. The first signal attributes a penalty proportional to the time gap between the package delivery and the requested time window. In contrast, the second signal simply penalises package deliveries outside the time window. Additionally, the penalties are the same whether the package was delivered early or late. Accordingly, serving the order 10 minutes earlier would yield the same punishment as 10 minutes late. Thus, the time gap can be defined as follows:

$$\text{early\_delivery\_gap}_a = \max(0, \text{start}_a - \text{delivery\_time}_a) \tag{4.25}$$

$$\text{late\_delivery\_gap}_a = \max(0, \text{delivery\_time}_a - \text{end}_a) \tag{4.26}$$

$$\text{time\_gap}_a = \text{early\_delivery\_gap}_a + \text{late\_delivery\_gap}_a \tag{4.27}$$

, where

- $start_a$ and $end_a$ are the start and end of the requested time window of the customer selected by action $a$.
- $delivery\_time_a$ corresponds to the time when the vehicle arrived at the customer selected by action $a$.

If the customer does not specify a starting time for its time window, the value of *early_delivery_gap* will be 0. Similarly, when the end time is not defined, the value of *late_delivery_gap* will be 0.

Using the defined time gap, the *Error Reward* is detailed in Equation 4.28. The time gap is divided by 2 hours to reduce this value's order of magnitude.

$$\text{Error Reward}_a = -\frac{\text{time\_gap}_a}{2*60*60} \tag{4.28}$$

Equation 4.29 depicts the *Inside or Outside Reward*. This reward penalises the agent for delivering packages outside the delivery window on two levels. The first level considers a difference of up to 10 minutes, with a smaller penalty of -0.5. More significant deviations from the time window result in a penalty score of -1.

$$\text{Inside or Outside Reward}_a = -\begin{cases} 0 & \text{, if time\_gap}_a = 0 \\ 0.5 & \text{, if time\_gap}_a \leq 10*60 \\ 1 & \text{, otherwise} \end{cases} \tag{4.29}$$

## 4.2 Performance Metrics

In this section, we discuss how to evaluate the performance of our approach. Firstly, we start with existing performance evaluation metrics in the literature that can be applied in multiple domains to assess the method's quality. In particular, we provide an overview of both Reinforcement Learning and Transfer Learning metrics. Then, we discuss what could be considered a performance measure in our specific domain, the VRP and its variants.

### 4.2.1 Reinforcement Learning Metrics

The most common performance evaluation metrics in RL algorithms are the received reward and the time spent training and / or testing the model.

The received reward can be observed and analysed from two different perspectives. Firstly, in regard to a single episode (as a function of the number of steps in that episode), it can illustrate how the rewards are dispersed through the episode. Secondly, analysing the cumulative reward obtained in each epoch (as a function of the number of epochs) can describe the policy evolution throughout the training. To analyse the cumulative reward, we could accumulate it across multiple episodes and epochs, yielding the *Accumulated Reward* or resetting that accumulator between episodes or epochs, yielding the *Received Rewards per Episode* or *Received Rewards per Epoch*, respectively. Both metrics produce the same results as the *Received Rewards*' absolute value corresponds to the *Accumulated Reward*'s slope.

On the other hand, the time spent training and / or testing the model reflects the computational power it requires and its efficiency. This metric usually requires that both models train in the same or equivalent hardware and software.

### 4.2.2 Transfer Learning Metrics

For ease of comparison with other projects, we started by compiling evaluation metrics already in use in the literature. Zhu et al. [160] group these performance metrics according to their perspective and main focus, as seen in Figure 4.1, and the TL ability they evaluate. According to that ability, the metrics are separated into two groups, mastery and generalisation. Mastery indicates the agent's level of proficiency in the target domain. Generalisation indicates the agent's ability to efficiently adapt to the target domain using the transferred knowledge from the source task. Additionally, some metrics can fit into both of these categories, depending on the choice of threshold. Figure 4.2 visually demonstrates how each of the following metrics was classified.

- *Jumpstart performance* [116, 160, 141]: the initial performance of the agent in the target domain.
- *Performance sensitivity* [160]: the performance variance when using different hyperparameter settings.
- *Necessary knowledge amount* [160]: the necessary amount of knowledge required to transfer.
- *Necessary knowledge quality* [160]: the necessary quality of the transferred knowledge.
- *Time to threshold* [116, 160]: number of steps necessary to reach the specified threshold. A low threshold measures generalisation, whereas a higher threshold measures mastery.
- *Performance with fixed training epochs* [160, 141]: agent's performance after a fixed number of steps. A low number of steps evaluates generalisation, whereas a higher number of steps evaluates mastery.
- *Asymptotic performance* [116, 160, 141]: the final performance of the agent in the target domain.
- *Accumulated rewards* [116, 160]: the total reward accumulated by the agent.
- *Transfer ratio* [116, 160]: the ratio between the agent's performance with and without transfer.

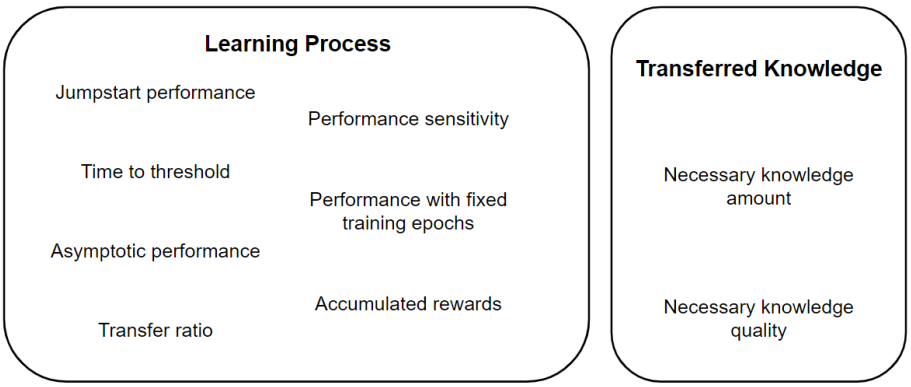


Figure 4.1: Result Evaluation Metrics divided according to their Perspective and Main Focus

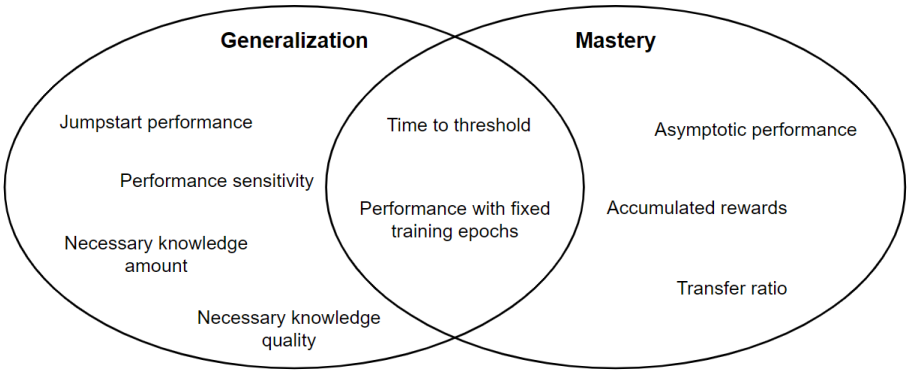


Figure 4.2: Result Evaluation Metrics divided into Mastery and Generalisation

### 4.2.3 Domain Metrics

As referred to in Section 3.1, when optimising the VRP, one can consider multiple conflicting objectives. While some of those objectives are not tangible, others are not so clear. In this project, we will focus more on the quantifiable ones, such as:

- Reducing the distance costs.
- Reducing the time costs.
- Improving the trip's energy efficiency.
- Reducing fuel consumption.
- Reducing carbon footprint.
- Reducing GHG emissions.
- Reducing $CO_2$ emissions.
- Reducing the number of vehicles needed.
- Reducing the number of deliveries made outside the customer's requested time window.
- Reducing the number of times a delivery is made earlier than requested.
- Reducing the number of times a delivery is made later than requested.
- Reducing the time gap between the time of the delivery and the requested time window.

Nonetheless, not all of the objectives presented are conflicting, and some are correlated. For example, fuel consumption is inherently connected to the carbon footprint and the emissions of GHG and $CO_2$. Moreover, by reducing fuel / energy consumption while still using the same type of vehicles and delivering all the packages to the customers, we are automatically increasing fuel

/ energy efficiency. Since there is a directly proportional relationship between fuel consumption and the carbon footprint, GHG, and $CO_2$ emissions, and an inversely proportional relationship between fuel consumption and fuel / energy efficiency, observing the fuel consumption metric should provide a good overview of how those metrics are performing. Therefore, one could consider the following performance measures:

- *Total Distance Travelled*: kilometres travelled by the fleet of vehicles.
- *Total Trip Time*: time spent outside the depot (travelling, serving customers, or waiting to serve customers) by the fleet of vehicles.
- *Number of Vehicles*: number of vehicles necessary to serve all the customers.
- *Total Fuel Consumption*: litres of fuel consumed by the vehicles.
- *Violated Time Windows*: number of deliveries made outside the requested time window.

It is essential to highlight that fuel consumption is still somewhat connected to the distance travelled or time spent on the trip. Despite that, it also depends on other factors, such as weight, speed, traffic congestion, and driver habits. According to [80], we can calculate GHG emissions using the formula:

$$GHG\ Emissions = D * W * EF \tag{4.30}$$

where,

- $D$ is the distance travelled by the vehicle.
- $W$ corresponds to the weight (both the vehicle's weight and the carried packages).
- $EF$ is the emissions factor. It corresponds to the amount of GHG emitted per litre of fuel used. This factor can be approximated as a constant.

Therefore, in this project, we will be observing and examining the following metrics:

- Unsatisfied Demand and Unsatisfied Customers
- Average Vehicle Load per Trip
- Total Distance Travelled
- Total Fuel Spent
- Percentage of Customers Served Outside their Requested Time Window (either Early or Late)

# Chapter 5

# Experimental Setup and Result Analysis

This chapter presents the datasets as well as the experiments, their configuration and parameters, and respective results and analysis. We start by introducing the datasets and their properties. Afterwards, we describe the general characteristics of the experiments. Then, we introduce each experiment and its specific settings, followed by the results and their analysis. To organise our experiments, we divided them based on the environment in which the agent operates and the method used. As a result, we conducted three experiments: the first two using Reinforcement Learning and the third one using Reinforcement Learning in conjunction with Transfer Learning. Each Reinforcement Learning experiment targeted a different environment: the first focused on the Capacitated Vehicle Routing Problem (CVRP), and the second on the Capacitated Vehicle Routing Problem with Soft Time Windows (CVRPSTW).

Some of the graphs in this chapter present relative values instead of absolute ones. These relative values are calculated within each run, meaning they were calculated using the formula:

$$relative\ value = \frac{value - minimum\ value\ of\ the\ run}{maximum\ value\ of\ the\ run - minimum\ value\ of\ the\ run} \tag{5.1}$$

While the absolute values allow us to compare the agent's performance, using relative values enables us to compare the evolution of the agent's performance. The use of relative values is necessary due to big disparities between absolute values or the impossibility of comparing absolute values.

The lines in the graphs represent the moving average of the last five values. Additionally, some graphs include a coloured area surrounding the lines delimited by the maximum and minimum values of the last five values. In the graphs with more lines, the coloured area was omitted for legibility.

For the complete and exact hyperparameters used in each of the experiments, see Appendix B.

## 5.1 Datasets

Before conducting the experiments, it is necessary to analyse the data that we will employ. The first one comes from the Loggi Benchmark for Urban Deliveries (LoggiBUD) [74], while the second one comes from the 2021 Amazon Last Mile Routing Research Challenge [81, 91]. From now on, we will refer to them as the Loggi dataset and the Amazon dataset, respectively. Note that both datasets only assume deliveries, excluding the concept of pickup points.

Some general properties for each dataset are presented in Table 5.1. Those properties include:

- *Dataset*: the name of the dataset.
- *N Cities*: the number of cities depicted in each dataset.
- *CVRP*: whether or not one can consider the CVRP when using this dataset.
- *CVRPSTW*: whether or not one can consider the CVRPSTW when using this dataset.

Table 5.1: Properties of the Datasets

| Dataset | N Cities | CVRP | CVRPSTW |
|---|---|---|---|
| **Loggi** | 3 | X | |
| **Amazon** | 5 | X | X |

Tables 5.2 and 5.3 describe each of the cities included in the Loggi and Amazon datasets, respectively. This characterisation includes:

- *City*: the city's name.
- *N Routes*: the number of routes that contain delivery points in that city.
- *N Points*: the total number of delivery points across all the available routes.
- *Min Points/Route*: the minimum number of delivery points per route in that city (excludes routes that do not deliver items to that city).
- *Max Points/Route*: the maximum number of delivery points per route in that city (excludes routes that do not deliver items to that city).
- *Avg Points/Route*: the average number of delivery points per route in that city (excludes routes that do not deliver items to that city).

The Loggi dataset contains 1320 CVRP instances simulating large-scale delivery problems within Belém, Brasília, and Rio de Janeiro, some of Brazil's largest cities. Even though Loggi proposes three increasingly complex tasks capable of being solved with this dataset, in this project, we exclusively focus on the first one, the CVRP. However, we introduce some modifications to the

optimisation objective. Instead of only minimising the total distance travelled, other optimisation needs, discussed in Section 4.2.3, are taken into account.

Table 5.2: Properties of Cities Present in the Dataset from Loggi [74]

| City | N Routes | N Points | Min Points/Route | Max Points/Route | Avg Points/Route |
|---|---|---|---|---|---|
| **Belém** | 240 | 533973 | 211 | 5108 | 2224.89 |
| **Brasília** | 360 | 1179407 | 765 | 5636 | 3276.13 |
| **Rio de Janeiro** | 720 | 3466092 | 1758 | 8421 | 4814.02 |

The Amazon dataset comprises 9177 historical routes. They contain delivery points scattered throughout Austin, Chicago, Seattle, Boston, and Los Angeles. While some of the delivery points include multiple packages, for ease of generalisation with the previous dataset, we treat each package as an individual delivery point when converting the routes to instances. Specifically, Table 5.3 comprises the number of delivery points after separating each package.

Table 5.3: Properties of Cities Present in the Dataset from Amazon [81]

| City | N Routes | N Points | Min Points/Route | Max Points/Route | Avg Points/Route |
|---|---|---|---|---|---|
| **Austin** | 335 | 79977 | 153 | 304 | 238.74 |
| **Boston** | 1345 | 309925 | 151 | 299 | 230.43 |
| **Seattle** | 1505 | 345343 | 151 | 299 | 229.46 |
| **Chicago** | 1472 | 372259 | 158 | 299 | 252.89 |
| **Los Angeles** | 4520 | 1077713 | 150 | 299 | 238.43 |

Both datasets only consider delivery points and disregard pickups. Additionally, they both assume a homogeneous fleet without specifying restrictions on the number of vehicles or the maximum number of trips per vehicle. As a result, in this project, we will consider a single vehicle capable of completing an infinite number of trips. This presumption is analogous to assuming an infinite number of vehicles, where each vehicle completes a single trip, except time accumulates between trips.

## 5.2 Core Configuration

Since our goals are divided into two groups, we present two separate pipelines for our experiments. When using the first pipeline, we explore the effects of different representations, action restrictions and reward functions on the RL's agent overall learning and performance. The second pipeline incorporates the first one and is intended to examine the effects of TL on the new agent's performance.

As previously mentioned in Section 2.1, an RL agent maps environment states to actions. The first pipeline, illustrated in Figure 5.1, corresponds to a single one of these mappings. It starts by deconstructing the environment state and organising it into nodes, edges, and their features.

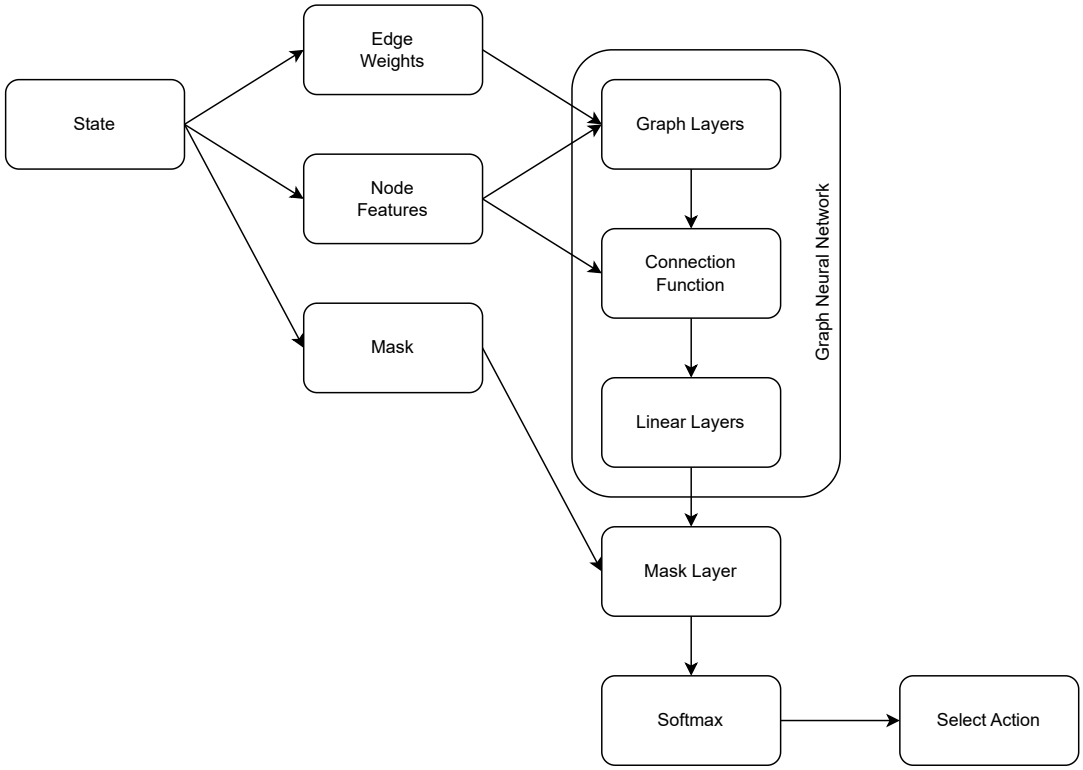


Figure 5.1: Steps Taken to Map an Environment State to an Action

Figure 5.3 exemplifies the second pipeline. This pipeline presents how Transfer Learning is applied to RL agents. Typically, an RL agent uses a dataset to train its model, as in Figure 5.2. When using Transfer Learning, the agent first receives knowledge from an already trained source model and then trains using an entirely new dataset.

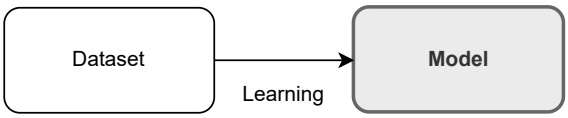


Figure 5.2: Reinforcement Learning Pipeline without Transfer Learning

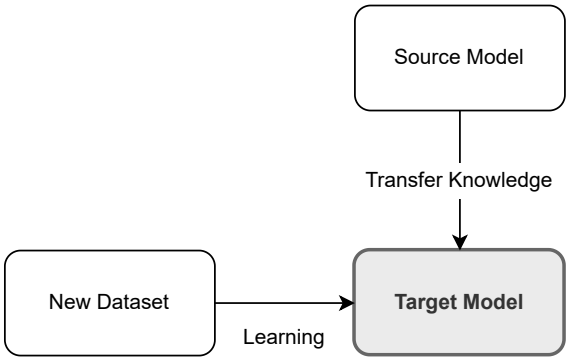


Figure 5.3: Transfer Learning Pipeline

After showcasing our pipelines, we proceed to provide their setup and configuration details. The REINFORCE algorithm described in Section 2.1.2 was used to train the agent. Due to time restrictions, optimising all the hyperparameters used was impossible. Therefore, the Graph Neural Network configuration is similar to the one presented in [39]. It contains a single Graph Convolutional Network layer, followed by a connection function and three Linear layers. The network weights are initialised as an orthogonal matrix, and its bias is initialised as zero.

Additionally, we defined the learning rate to be $10^{-4}$ since the authors of [56] deemed it the most appropriate. Furthermore, we performed some experiments regarding the discount factor ($\gamma$) value, which can be seen in Figure 5.4. We used a discount factor of 0.98 for all our other experiments, as it provided more stable and convergent learning. Aside from that, we went with the default parameters considered by the library we were using [133].

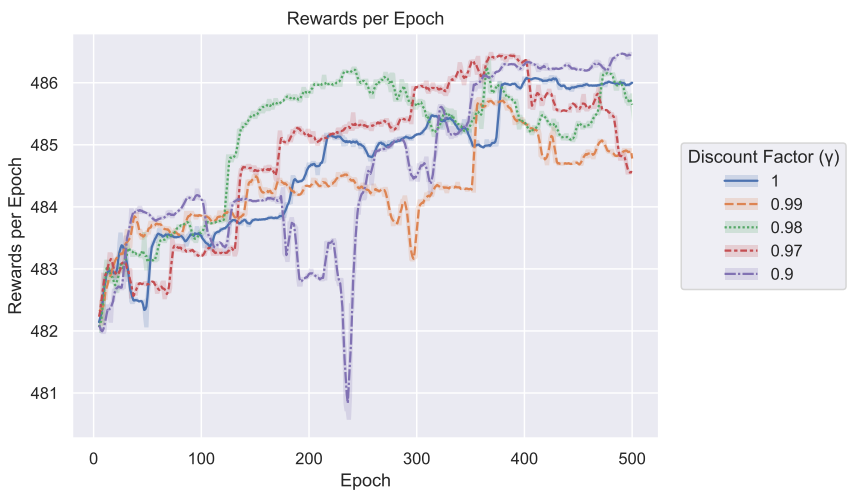


Figure 5.4: Discount Factor Tuning: Rewards per Episode

The exact hyperparameters used in each experiment are described in full detail in Appendix B.

## 5.3 Capacitated Vehicle Routing Problem (CVRP)

The CVRP environment deals with the Capacitates Vehicle Routing Problem scenario. As such, it is subject to its constraints. Additionally, from the node features and reward signals recognised in Sections 4.1.1 and 4.1.3, respectively, it only employs the ones related to the general routing or the vehicle capacity constraint. Moreover, we only examine domain metrics associated with routing and vehicle capacity from the ones presented in Section 4.2.3.

The reward functions approached in this section are detailed below.

$$\textbf{Distance}_a = \text{Distance Reward}_a \tag{5.2}$$

$$\textbf{Distance Norm.}_a = \text{Normalised Distance Reward}_a + \text{Step Reward}_a \tag{5.3}$$

$$\textbf{Fraction}_a = \text{Fraction Reward}_a + \text{Step Reward}_a \tag{5.4}$$

$$\textbf{Savings}_a = \text{Savings Reward}_a \tag{5.5}$$

$$\textbf{Fuel Cons.}_a = \text{Fuel Consumption Reward}_a + \text{Step Reward}_a \tag{5.6}$$

$$\textbf{Multi (D + F)}_a = \text{Normalised Distance Reward}_a + \text{Fuel Consumption Reward}_a + \text{Step Reward}_a \tag{5.7}$$

$$\textbf{Sparse (Dist. Norm.)} = \begin{cases} \sum_a^{a \in A} (\text{Distance Norm.}_a) & \text{if episode is over} \\ 0 & \text{, otherwise} \end{cases} \tag{5.8}$$

, where $A$ represents the sequence of actions executed during the episode.

The last equation represents the *Sparse Reward*. Identical to the *Distance Norm.* in terms of received rewards per episode, it only provides the rewards to the agent at the end of the episode instead of at every step.

### 5.3.1 The Masking Mechanism

Due to the difficulties in guiding the agent to a feasible solution referenced in Section 4.1.2, we implemented a mask that prevents selecting nodes that lead to an invalid sequence of actions. This section displays the experimental results that verify that using this specific mask guarantees a better jumpstart and overall performance. Thus, we compare the agent's learning when using different masking procedures. The first agent (*None*) will be operating without any restriction. The second agent (*Noop*) will use a simpler mask to avoid not moving (selecting the node the vehicle is currently in). The third agent (*Invalid*) will use the mask described in Section 4.1.2. Remembering, this mask blocks our agent from deciding to execute actions that lead to unfeasible or less-than-optimal solutions, such as:

- Selecting the current node.

- Selecting a customer who has already been served (whose demand is 0).

- Selecting a customer the vehicle cannot serve (its demand exceeds the remaining capacity of the vehicle).

Since agents *None* and *Noop* are not guaranteed to finish the episode, a maximum number of steps for a single episode was established. In the event that the agent surpassed it, the episode would stop, and a penalty of 1000 would be issued. The *Invalid* agent cannot visit the same customer twice. As such, there is no need to establish an upper bound on the number of actions it can execute per episode.

Figure 5.5 depicts the absolute values of the total rewards received per episode. While this figure does not allow us to evaluate the evolution of rewards throughout learning, it highlights the difference between the performance of the various agents.

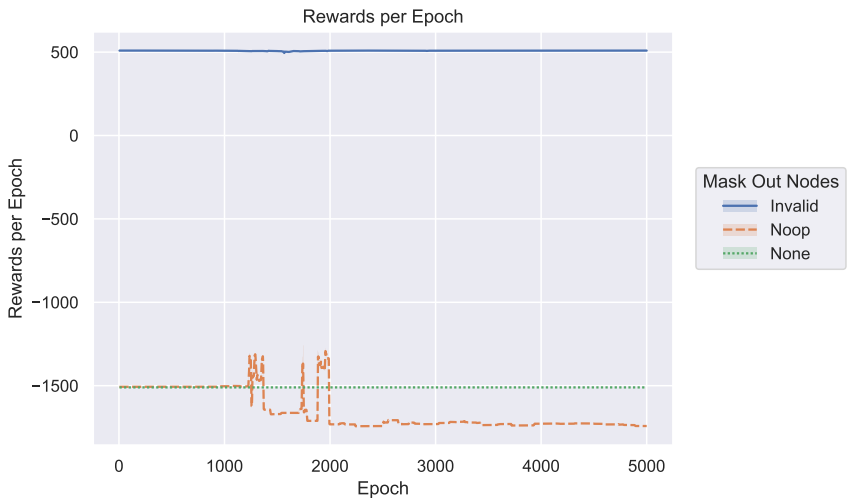


Figure 5.5: Rewards per Episode Comparison between Agents using Different Masks (Absolute Values)

Figure 5.6 illustrates the evolution of rewards received per epoch using relative values. These normalised relative values allow us to perceive the evolution of the rewards and analyse the convergence. These relative values do not contain information about which agent performs better but rather which agent has a better or worse evolution. A downward curve indicates that the agent's performance degrades as it trains more, and an upwards curve demonstrates that its performance improves over time.

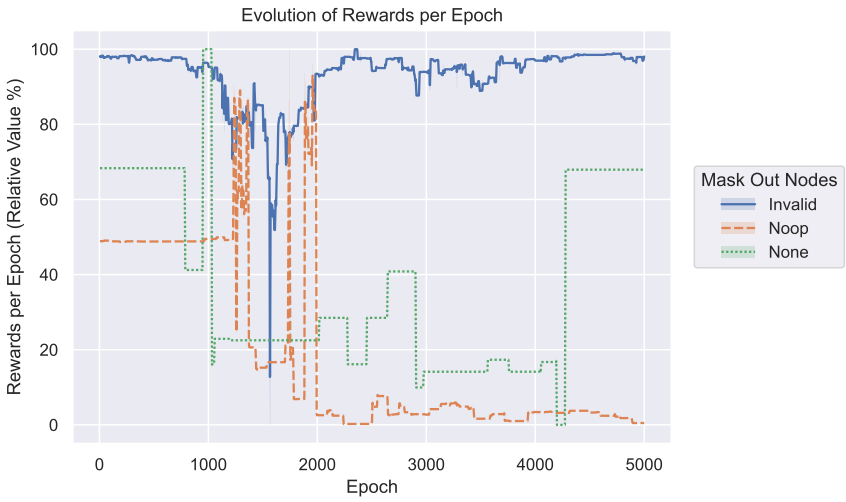


Figure 5.6: Rewards per Episode Comparison between Agents using Different Masks (Relative Values)

As can be observed in the previous two figures, the agent using *Invalid* clearly outperforms the others. The size of the action space, combined with the fact that this problem is combinatorial, difficults the learning of the other two agents. These other agents have to go through a lot of the search space in order to figure out what was initially imposed on the *Invalid* agent. Since this agent sees its search space reduced at each step, it will have fewer problems finishing all the deliveries and the episode. On top of that, it will avoid the penalty applied for not being able to complete all the deliveries.

Intuitively, by not having to explore a large group of invalid actions and negative rewards, the agent will have more opportunities to explore valid actions. Indeed, since the agent starts without any prior knowledge of which actions are valid or invalid, it needs to explore the entirety of the action space to determine which actions are beneficial or detrimental before choosing which actions are better.

Figures 5.7, 5.8 and 5.9 demonstrate the problems with the solutions the *Noop* and *None* agents found. These agents get lost on local minimums that do not produce feasible solutions.

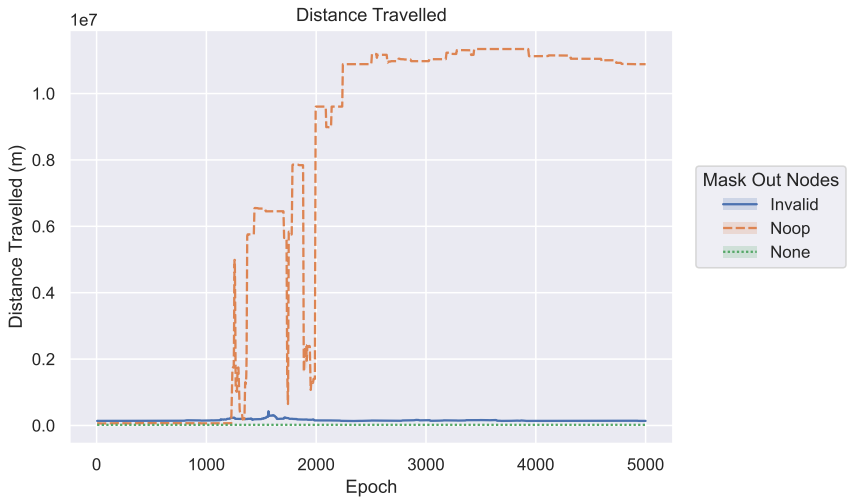


Figure 5.7: Distance Travelled Comparison between Agents using Different Masks (Absolute Values)

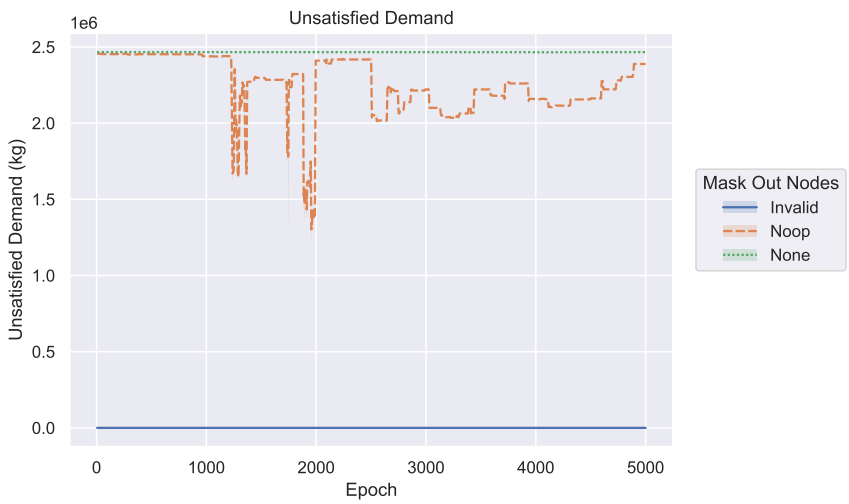


Figure 5.8: Unsatisfied Demand Comparison between Agents using Different Masks (Absolute Values)

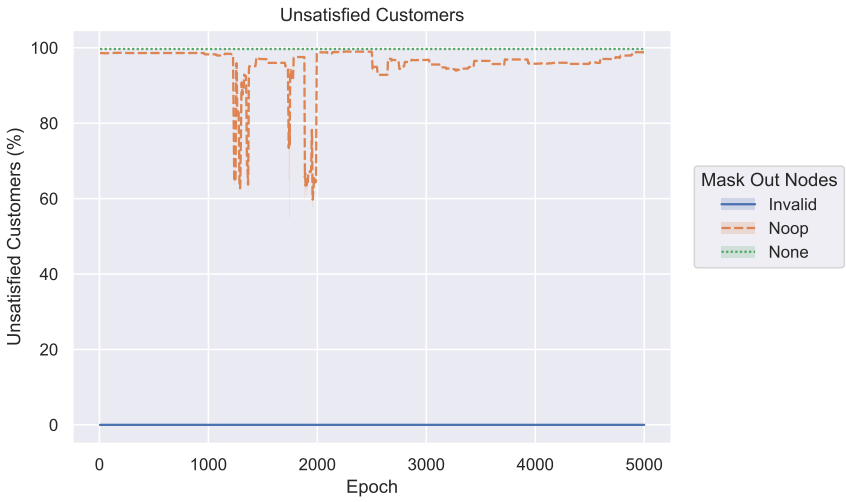


Figure 5.9: Unsatisfied Customers (%) Comparison between Agents using Different Masks (Absolute Values)

While agent *Invalid* serves all the customers, agents *Noop* and *None* do not. These two wander around almost aimlessly without serving almost anyone. In fact, they both leave most customers unserved, with *Noop* averaging 95.81% and *None* at 99.68%. While the agent *Noop* manages to serve 45.24% of customers during epoch 1744 (leaving 54.76% of customers unserved), the agent without any restrictions (*None*) never serves more than 0.32% of customers.

*None* opts for not leaving the depot or simply visiting a single customer and then stopping there for the rest of the epoch. On the other hand, since *Noop* cannot remain in the same location, it usually hops in a loop between 2 or 3 customers, occasionally breaking the circle to serve a few customers or engage in a new loop. During their best epochs (947 and 1744, respectively), agents *None* and *Noop* serve 4 and 258 customers, respectively, out of 1293. Additionally, they visit the depot 512 and 259 times and repeat customers 2044 and 2043 times.

### 5.3.2 Node Features

This section evaluates the effects of different node features on the agent's performance. For that, we select some of the referred properties as obligatory and some as optional.

Table 5.4 identifies all the node features considered and whether they were obligatory or optional.

Table 5.4: Node Features Considered in the *CVRP* Experiments

| **Node Feature** | **Obligatory** | **Optional** |
|---|---|---|
| *location* | X | |
| *depot* | X | |
| *demand* | X | |
| *current_capacity* | | X |
| *next_capacity* | | X |
| *possible_capacity* | | X |
| *can_serve* | | X |
| *fuel_consumption* | | X |
| *cost* | | X |
| *mask* | | X |

The results of this experiment can be seen in Table 5.5. For simplicity, only the optional node features are presented in the table. The column *Accumulated Rewards* holds the cumulative rewards obtained through 25 training epochs. The best performant agents attain good accumulated rewards, as well as good initial and final performances. Additionally, the initial and final performances should be different to indicate the potential to develop and evolve over time. Figure 5.10 describes the distribution of the cumulative rewards obtained per agent.

Table 5.5: Optional Node Features Considered and Respective Accumulated Rewards in the *CVRP* Environment

| Node Features | | | | | | | Accumulated | Start | End |
|---|---|---|---|---|---|---|---|---|---|
| *current_capacity* | *next_capacity* | *possible_capacity* | *can_serve* | *fuel_consumption* | *cost* | *mask* | Rewards | Reward | Reward |
| | | | | | | | 11999.74 | 480.34 | 480.37 |
| X | | | | | | | 12230.2 | 489.17 | 489.07 |
| | X | | | | | | 12224.98 | 488.96 | 488.99 |
| | | X | | | | | 12224.17 | 488.96 | 489.05 |
| | | | X | | | | 1512.24 | 60.49 | 60.49 |
| | | | | X | | | 2372.31 | 94.38 | 94.72 |
| | | | | | X | | 1512.24 | 60.49 | 60.49 |
| | | | | | | X | 1512.24 | 60.49 | 60.49 |
| X | X | | | | | | 12229.04 | 489.15 | 488.93 |
| X | | X | | | | | 12199.93 | 488.21 | 487.91 |
| X | | | X | | | | 12200.51 | 488.04 | 488.13 |
| X | | | | X | | | 11748.39 | 469.7 | 470.01 |
| X | | | | | X | | 12189.23 | 487.45 | 487.6 |
| X | | | | | | X | 12200.03 | 488.04 | 488.04 |
| | X | X | | | | | 12213.9 | 488.3 | 488.59 |
| | X | | X | | | | 12202.52 | 488.09 | 488.2 |
| | X | | | X | | | 11792.51 | 473.25 | 469.78 |
| | X | | | | X | | 12195.44 | 487.85 | 487.8 |
| | X | | | | | X | 12200.77 | 488.09 | 488.03 |
| | | X | X | | | | 12249.6 | 489.96 | 490.07 |
| | | X | | X | | | 11755.12 | 469.95 | 469.83 |
| | | X | | | X | | 12220.0 | 488.73 | 488.88 |
| | | X | | | | X | 12251.6 | 489.98 | 490.04 |
| | | | X | X | | | 11810.72 | 472.38 | 472.53 |
| | | | X | | X | | 12267.65 | 492.78 | 490.16 |
| | | | X | | | X | 11763.41 | 471.03 | 470.78 |
| | | | | X | X | | 12721.96 | 508.87 | 508.88 |

Table 5.5: Optional Node Features Considered and Respective Accumulated Rewards in the *CVRP* Environment

| Node Features | | | | | | | Accumulated | Start | End |
|---|---|---|---|---|---|---|---|---|---|
| *current_capacity* | *next_capacity* | *possible_capacity* | *can_serve* | *fuel_consumption* | *cost* | *mask* | Rewards | Reward | Reward |
| | | | | X | | X | 12721.4 | 508.87 | 508.9 |
| | | | | | X | X | 12680.53 | 507.05 | 507.03 |
| X | X | X | | | | | 12261.81 | 490.23 | 490.72 |
| X | X | | X | | | | 10730.7 | 432.4 | 428.81 |
| X | X | | | X | | | 12719.69 | 508.92 | 508.66 |
| X | X | | | | X | | 10847.17 | 438.59 | 433.33 |
| X | X | | | | | X | 11605.68 | 432.4 | 439.29 |
| X | | X | X | | | | 10645.22 | 425.22 | 426.26 |
| X | | X | | X | | | 12708.55 | 508.45 | 508.4 |
| X | | X | | | X | | 10893.33 | 432.54 | 465.49 |
| X | | X | | | | X | 10652.52 | 425.22 | 425.99 |
| X | | | X | X | | | 12540.71 | 500.67 | 501.9 |
| X | | | X | | X | | 12560.6 | 502.42 | 502.43 |
| X | | | X | | | X | 12064.27 | 482.35 | 482.51 |
| X | | | | X | X | | 2819.84 | 81.02 | 118.5 |
| X | | | | X | | X | 2593.0 | 80.72 | 179.99 |
| X | | | | | X | X | 2820.73 | 112.58 | 112.81 |
| | X | X | X | | | | 10485.09 | 420.12 | 419.8 |
| | X | X | | X | | | 12719.44 | 508.77 | 508.83 |
| | X | X | | | X | | 11827.72 | 427.1 | 491.82 |
| | X | X | | | | X | 10440.88 | 420.13 | 417.87 |
| | X | | X | X | | | 12629.0 | 505.6 | 503.91 |
| | X | | X | | X | | 12560.51 | 502.45 | 502.48 |
| | X | | X | | | X | 12064.28 | 482.34 | 482.47 |
| | X | | | X | X | | 2145.14 | 81.07 | 78.6 |
| | X | | | X | | X | 3707.98 | 80.9 | 174.9 |
| | X | | | | X | X | 2820.91 | 112.59 | 112.81 |

Table 5.5: Optional Node Features Considered and Respective Accumulated Rewards in the *CVRP* Environment

| Node Features | | | | | | | Accumulated | Start | End |
|---|---|---|---|---|---|---|---|---|---|
| *current_capacity* | *next_capacity* | *possible_capacity* | *can_serve* | *fuel_consumption* | *cost* | *mask* | Rewards | Reward | Reward |
| | | X | X | X | | | 12539.63 | 500.14 | 501.44 |
| | | X | X | | X | | 12553.47 | 502.18 | 502.05 |
| | | X | X | | | X | 12063.52 | 482.25 | 482.3 |
| | | X | | X | X | | 1875.5 | 80.04 | 73.47 |
| | | X | | X | | X | 1921.27 | 79.75 | 72.25 |
| | | X | | | X | X | 11756.36 | 470.21 | 470.24 |
| | | | X | X | X | | 3056.8 | 176.33 | 115.86 |
| | | | X | X | | X | 3452.53 | 175.68 | 132.86 |
| | | | X | | X | X | 1516.9 | 60.68 | 60.68 |
| | | | | X | X | X | 12727.81 | 509.19 | 509.05 |
| X | X | X | X | | | | 1512.24 | 60.49 | 60.49 |
| X | X | X | | X | | | 12604.59 | 502.98 | 506.61 |
| X | X | X | | | X | | 1512.24 | 60.49 | 60.49 |
| X | X | X | | | | X | 1512.24 | 60.49 | 60.49 |
| X | X | | X | X | | | 12664.32 | 507.15 | 507.14 |
| X | X | | X | | X | | 1512.24 | 60.49 | 60.49 |
| X | X | | X | | | X | 1512.24 | 60.49 | 60.49 |
| X | X | | | X | X | | 12727.21 | 509.06 | 509.13 |
| X | X | | | X | | X | 12724.55 | 509.06 | 509.07 |
| X | X | | | | X | X | 1512.24 | 60.49 | 60.49 |
| X | | X | X | X | | | 12601.66 | 502.47 | 506.84 |
| X | | X | X | | X | | 1512.24 | 60.49 | 60.49 |
| X | | X | X | | | X | 1512.24 | 60.49 | 60.49 |
| X | | X | | X | X | | 12725.51 | 509.04 | 508.98 |
| X | | X | | X | | X | 12725.52 | 509.04 | 509.01 |
| X | | X | | | X | X | 1512.24 | 60.49 | 60.49 |
| X | | | X | X | X | | 12726.56 | 509.09 | 509.03 |

Table 5.5: Optional Node Features Considered and Respective Accumulated Rewards in the *CVRP* Environment

| Node Features | | | | | | | Accumulated | Start | End |
|---|---|---|---|---|---|---|---|---|---|
| *current_capacity* | *next_capacity* | *possible_capacity* | *can_serve* | *fuel_consumption* | *cost* | *mask* | Rewards | Reward | Reward |
| X | | | X | X | | X | 12726.92 | 509.09 | 509.07 |
| X | | | X | | X | X | 12483.38 | 499.7 | 499.38 |
| X | | | | X | X | X | 12681.06 | 508.12 | 506.56 |
| | X | X | X | X | | | 12233.62 | 480.89 | 491.79 |
| | X | X | X | | X | | 1512.24 | 60.49 | 60.49 |
| | X | X | X | | | X | 1512.24 | 60.49 | 60.49 |
| | X | X | | X | X | | 12727.1 | 509.02 | 509.05 |
| | X | X | | X | | X | 12727.56 | 509.02 | 509.11 |
| | X | X | | | X | X | 2157.15 | 60.49 | 125.04 |
| | X | | X | X | X | | 12727.17 | 509.07 | 509.1 |
| | X | | X | X | | X | 12727.05 | 509.07 | 509.09 |
| | X | | X | | X | X | 12453.78 | 497.98 | 498.56 |
| | X | | | X | X | X | 12667.49 | 506.68 | 506.53 |
| | | X | X | X | X | | 12729.05 | 509.32 | 509.09 |
| | | X | X | X | | X | 12729.57 | 509.32 | 509.05 |
| | | X | X | | X | X | 12500.58 | 500.37 | 499.98 |
| | | X | | X | X | X | 12687.85 | 507.94 | 507.69 |
| | | | X | X | X | X | 12356.65 | 494.48 | 493.25 |
| X | X | X | X | X | | | 12250.43 | 493.22 | 483.78 |
| X | X | X | X | | X | | 12234.09 | 489.33 | 489.44 |
| X | X | X | X | | | X | 12231.52 | 489.06 | 489.29 |
| X | X | X | | X | X | | 12433.32 | 497.95 | 496.57 |
| X | X | X | | X | | X | 12404.27 | 497.94 | 494.05 |
| X | X | X | | | X | X | 12238.3 | 489.44 | 489.59 |
| X | X | | X | X | X | | 12719.96 | 508.68 | 508.89 |
| X | X | | X | X | | X | 12718.18 | 508.68 | 508.73 |
| X | X | | X | | X | X | 12351.61 | 494.17 | 494.32 |

Table 5.5: Optional Node Features Considered and Respective Accumulated Rewards in the *CVRP* Environment

| Node Features | | | | | | | Accumulated | Start | End |
|---|---|---|---|---|---|---|---|---|---|
| *current_capacity* | *next_capacity* | *possible_capacity* | *can_serve* | *fuel_consumption* | *cost* | *mask* | Rewards | Reward | Reward |
| X | X | | | X | X | X | 8339.61 | 333.56 | 333.58 |
| X | | X | X | X | X | | 12714.82 | 508.37 | 508.62 |
| X | | X | X | X | | X | 12711.09 | 508.4 | 508.46 |
| X | | X | X | | X | X | 12385.96 | 496.37 | 495.12 |
| X | | X | | X | X | X | 9183.49 | 369.79 | 332.68 |
| X | | | X | X | X | X | 11749.56 | 471.76 | 468.55 |
| | X | X | X | X | X | | 12714.47 | 508.2 | 508.73 |
| | X | X | X | X | | X | 12689.54 | 508.22 | 507.01 |
| | X | X | X | | X | X | 12433.09 | 497.54 | 497.16 |
| | X | X | | X | X | X | 9228.72 | 368.69 | 369.23 |
| | X | | X | X | X | X | 11165.22 | 449.75 | 444.09 |
| | | X | X | X | X | X | 11791.94 | 472.52 | 471.3 |
| X | X | X | X | X | X | | 11732.67 | 470.65 | 467.52 |
| X | X | X | X | X | | X | 11650.85 | 470.43 | 467.7 |
| X | X | X | X | | X | X | 12211.47 | 487.89 | 488.86 |
| X | X | X | | X | X | X | 11378.99 | 478.64 | 433.14 |
| X | X | | X | X | X | X | 12186.4 | 488.99 | 487.66 |
| X | | X | X | X | X | X | 12418.77 | 498.02 | 496.88 |
| | X | X | X | X | X | X | 12385.74 | 492.98 | 496.48 |
| X | X | X | X | X | X | X | 12685.48 | 507.5 | 507.57 |

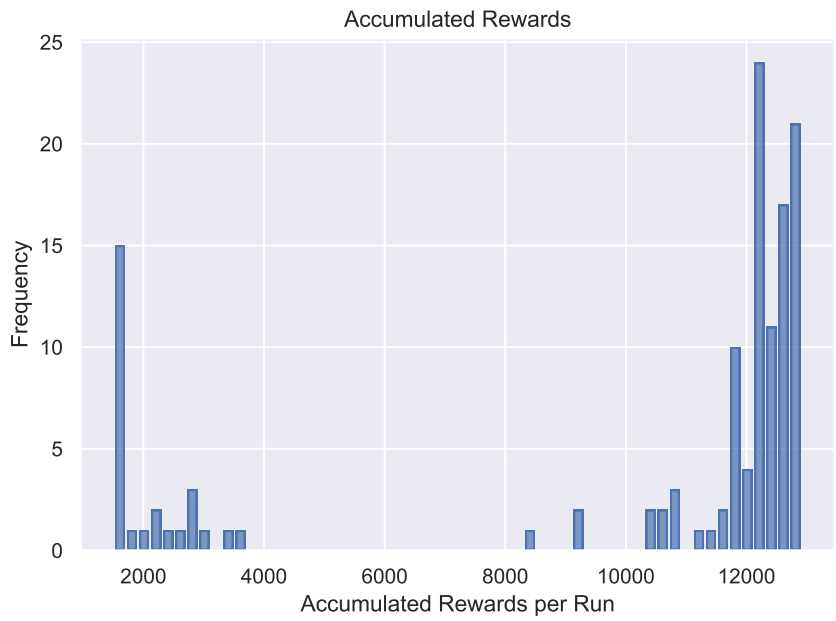


Figure 5.10: Distribution of Accumulated Rewards Obtained by Agents using Different Node Features through the Span of 25 Epochs

As can be seen, there is a significant variance in the accumulated rewards caused by using various node feature combinations. Additionally, Figure 5.11 showcases the effects of different node feature combinations throughout the 25 training epochs.

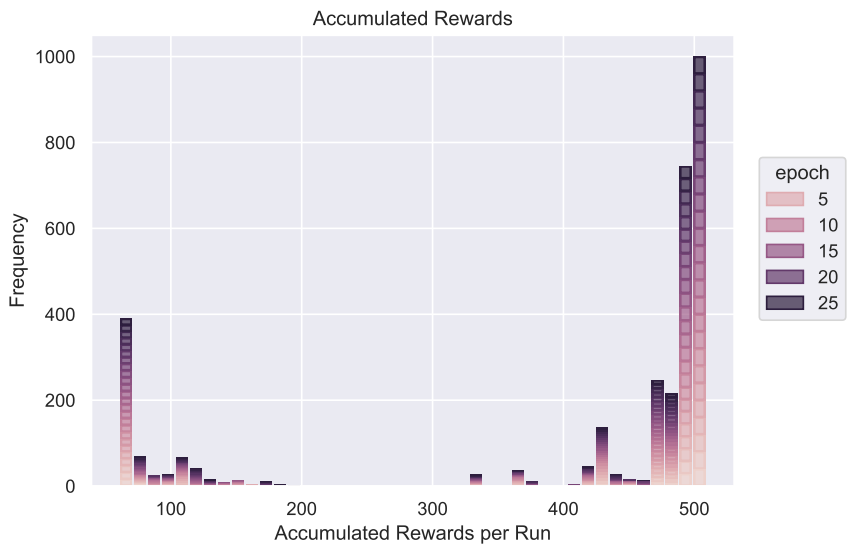


Figure 5.11: Distribution of Rewards per Epoch Obtained by Agents using Different Node Feature Combinations

This figure also observes a large range of values, indicating that node feature combinations

influence the initial and final performances of the agent as well as its training. This impact can be partly justified by the low amount of training completed by the agents. Since the agents in this experiment only train for 25 epochs, their performance is highly dependent on the network initialisation, which is an orthogonal matrix.

We decided to explore further the combination: *location*, *depot*, *demand*, *current_capacity*, *next_capacity*, *fuel_consumption*, and *cost* for the experiments regarding the *CVRP* environment.

### 5.3.3 Reward Functions

In order to prove the impact and importance of correctly designed reward functions, we performed two different experiments. Due to the nature of this section and to allow a fair comparison between agents, the absolute rewards per episode will not be considered. The rewards the agent receives will be inherently different as a result of different reward functions. Therefore, all the comparisons will regard only the relative rewards per episode and domain metrics.

In the first experiment, we use the mask addressed as *Invalid* in the previous section, which masks out invalid nodes, to test the agent's performance under different reward functions. Figures 5.12, 5.13, and 5.14 depict the effects of using different reward functions with the same objective on the agent's performance.

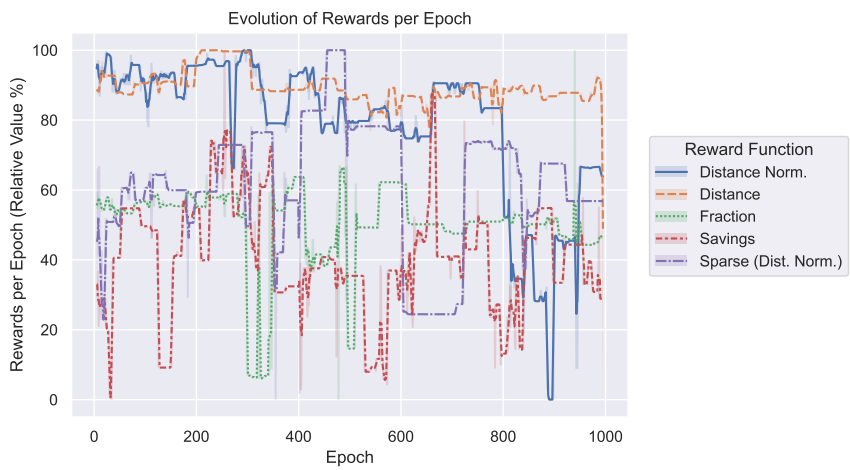


Figure 5.12: Rewards per Episode Comparison between Agents using Different Reward Functions (Relative Values)

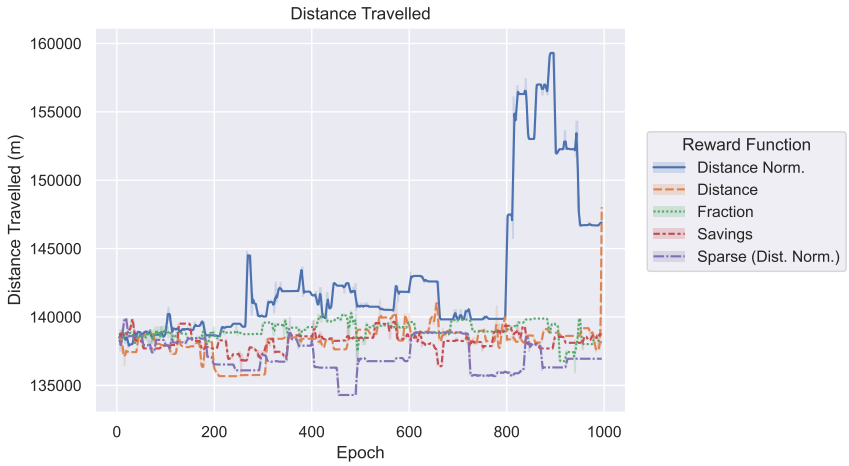


Figure 5.13: Distance Travelled Comparison between Agents using Different Masks (Absolute Values)

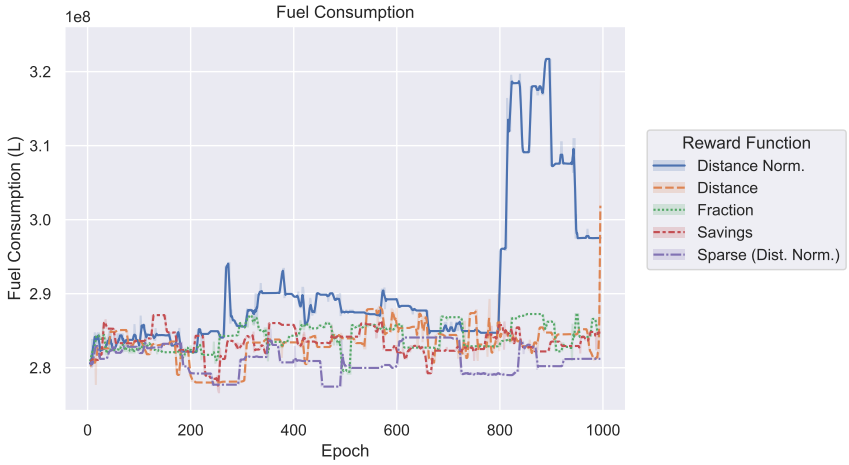


Figure 5.14: Fuel Consumption Comparison between Agents using Different Reward Functions (Absolute Values)

As expected, even though the agent's performance differs as a result of different reward functions, it still performs quite similarly when optimising the same objectives. Additionally, rewards such as *Fraction* and *Savings* note a bigger stability than the *Distance Norm.* reward. Surprisingly, the agent subject to *Sparse (Dist. Norm.)* performs very well. This is in great part due to the masking mechanism used. Since the mask guides the agent to finish the episode, when using the *Sparse (Dist. Norm.)* reward function, the agent will still receive the entirety of the episode rewards. While they are delayed, all the partial trips that compose the vehicle's route are considered and accounted for in the final reward. However, in the case that they do not use masking, the agent is not guaranteed to finish the episode. As such, the rewards received may vary, which will lead to very different results.

Figures 5.15, 5.16, and 5.17 portray the effects of using different reward functions attempting to optimise different objectives.

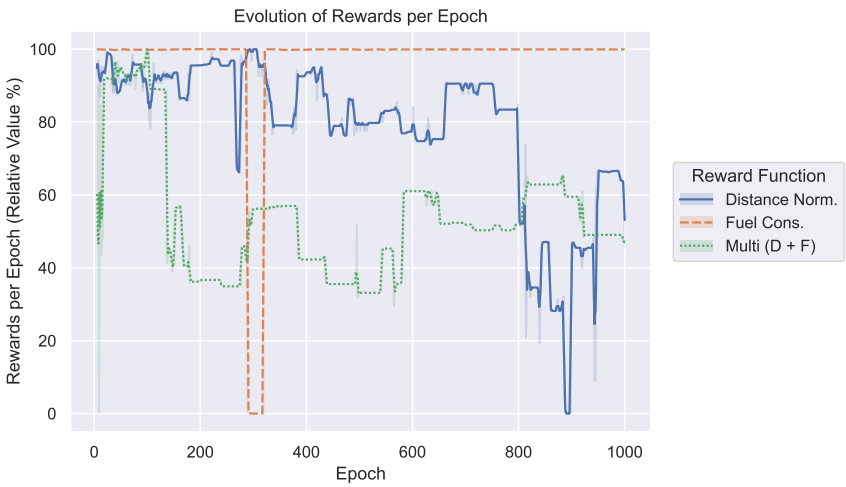


Figure 5.15: Rewards per Episode Comparison between Agents using Different Reward Functions (Relative Values)

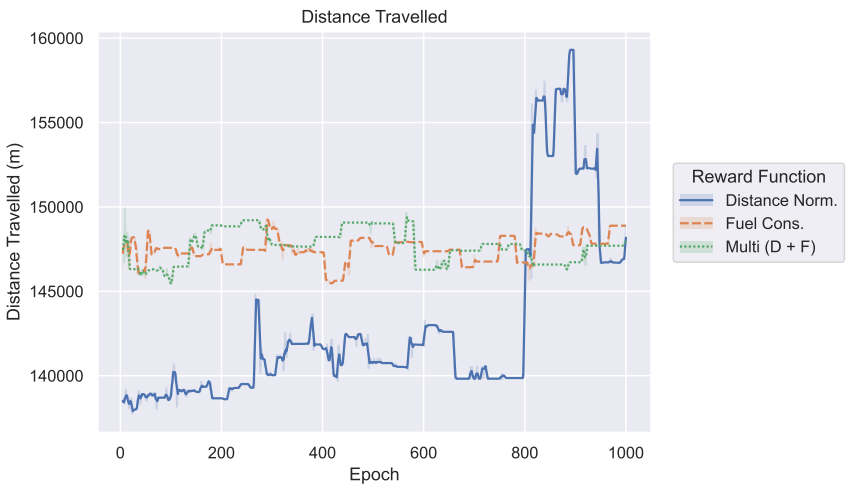


Figure 5.16: Unsatisfied Customers Comparison between Agents using Different Masks (Absolute Values)

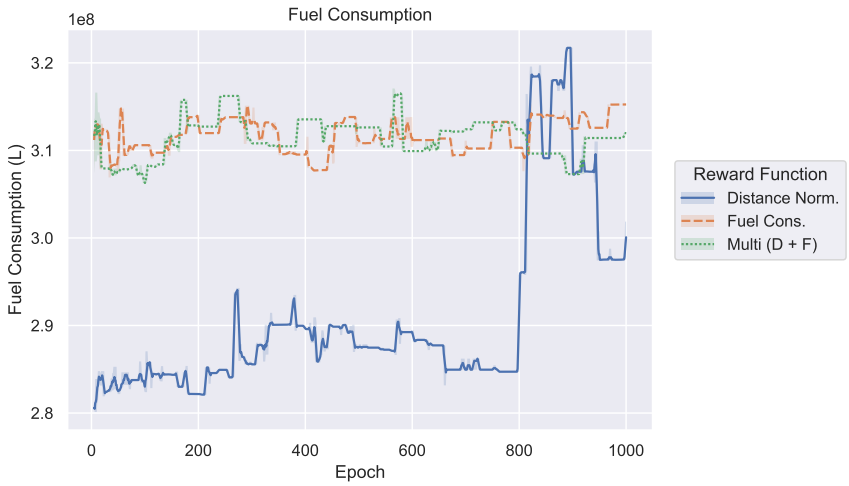


Figure 5.17: Fuel Consumption Comparison between Agents using Different Reward Functions (Absolute Values)

Most importantly, the behaviour is different when trying to optimise different objectives. Quite unexpectedly, the agent attempting to minimise the distance (using the *Distance Norm.* reward) outclasses its counterparts which consider fuel consumption minimisation in both distance travelled and fuel consumption.

There are two main factors which contribute to this underperformance. The first factor is the network initialisation does not favour this reward function. The node features were also chosen based on how well the agent performed with them when receiving rewards from the *Distance Norm.* function. On the other hand, the agent might have problems with credit assignment since the package must travel from the depot to the customer, and all that distance needs to be considered when calculating the fuel consumption. As such, delivering the heavier packages first and the lighter ones later leads to a much better reward than the opposite. Nonetheless, this is implicit and, therefore, not explicitly represented in the reward function. Thus, the agent might need more time and training to understand these caveats effectively.

In a second experiment, we experimented without the mask and effectively compared dense and sparse reward functions. In this scenario, we can visualise and interpret the absolute reward per episode since, when the actions executed are equal, both functions grant the same reward per episode. The only difference is using a dense reward function, *Distance Norm.*, for the *Dense* agent and a sparse reward function, *Sparse (Dist. Norm.)*, for the *Sparse* agent.

However, the agents require much more training when training without the mask. Due to time constraints, the experiment concluded after only 1000 epochs, which is not enough to preview the full picture of training with these two reward functions, as seen in Figure 5.18.

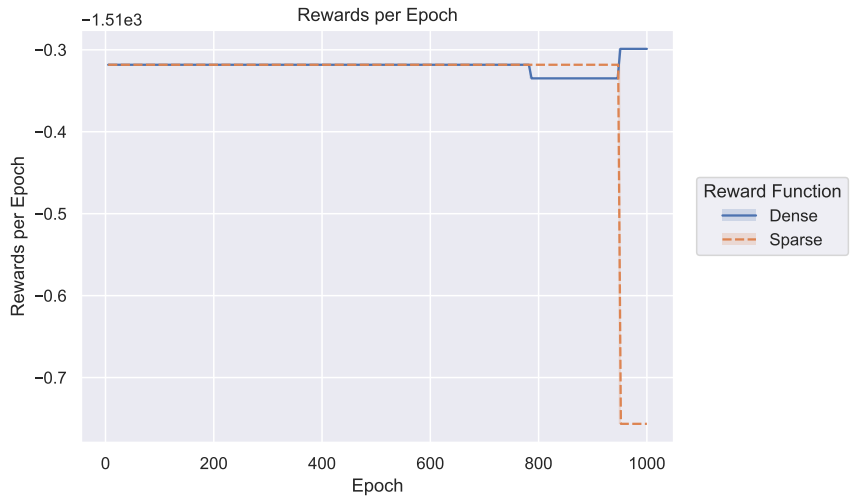


Figure 5.18: Rewards per Episode Comparison between Agents using Different Reward Functions (Absolute Values)

## 5.4 CVRP with Soft Time Windows (CVRPSTW)

The CVRPSTW environment represents the Capacitated Vehicle Routing Problem with Time Windows. Therefore, alongside the restrictions already presented in the previous section, the customers should be visited within a specified time window. As such, all the constraints introduced in Section 4.1 need to be respected. Moreover, any of the node features, reward signals, and performance metrics discussed in Sections 4.1.1, 4.1.3 and 4.2.3 is applicable.

Similarly to the previous section, we establish the reward functions as a linear combination of reward signals proposed. However, instead of testing the reward signals related to routing and the capacity constraints, we explore the ones related to the customer time windows. In this experiment, we explore two particular reward functions: the *TW Failed (D)* and the *TW Error (D)* described in Equations 5.9 and 5.10.

$$\textbf{TW Failed (D)}_a = \text{Distance Norm.}_a + \text{Inside or Outside Reward}_a \tag{5.9}$$

$$\textbf{TW Error (D)}_a = \text{Distance Norm.}_a + \text{Error Reward}_a \tag{5.10}$$

### 5.4.1 CVRP vs CVRPSTW

Before diving into the specifics of the CVRPSTW, it is important to understand the behavioural differences between agents operating in the CVRP and the CVRPSTW environments. Figures 5.19, 5.20,

and 5.21 expose the main differences between agents trained in the CVRP and CVRPSTW environments.

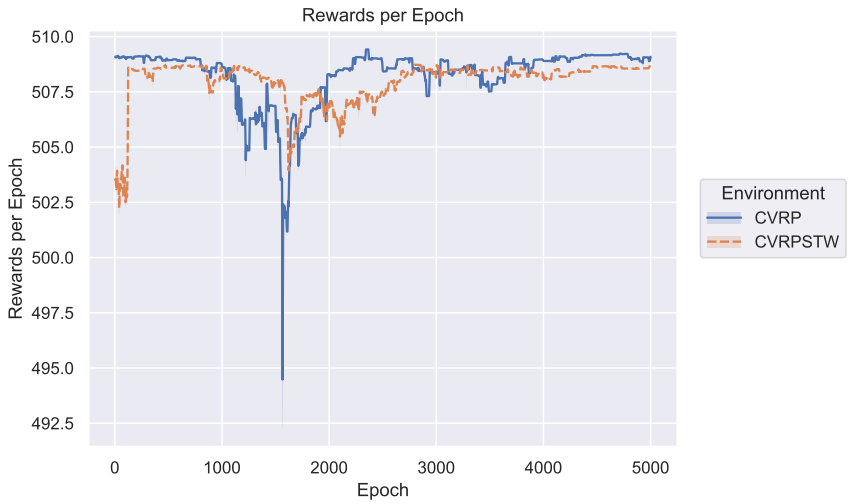


Figure 5.19: Rewards per Episode Comparison between Agents operating in Different Environments (Absolute Values)

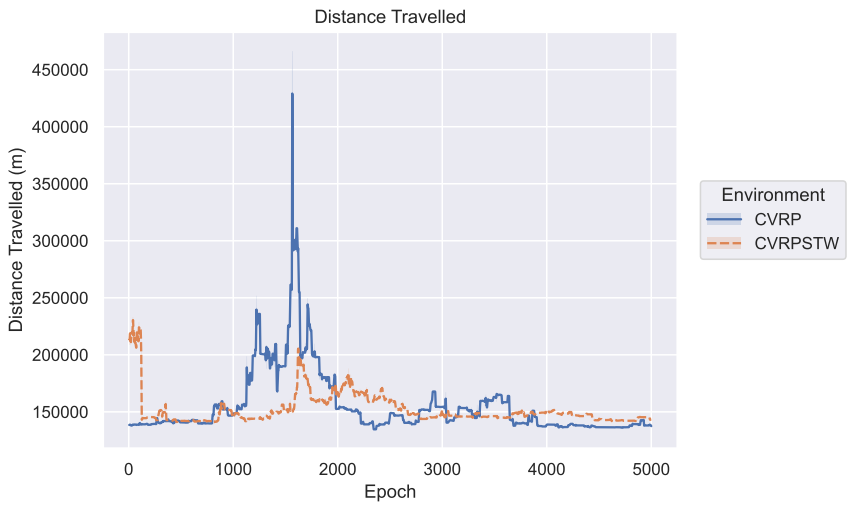


Figure 5.20: Distance Travelled Comparison between Agents operating in Different Environments (Absolute Values)

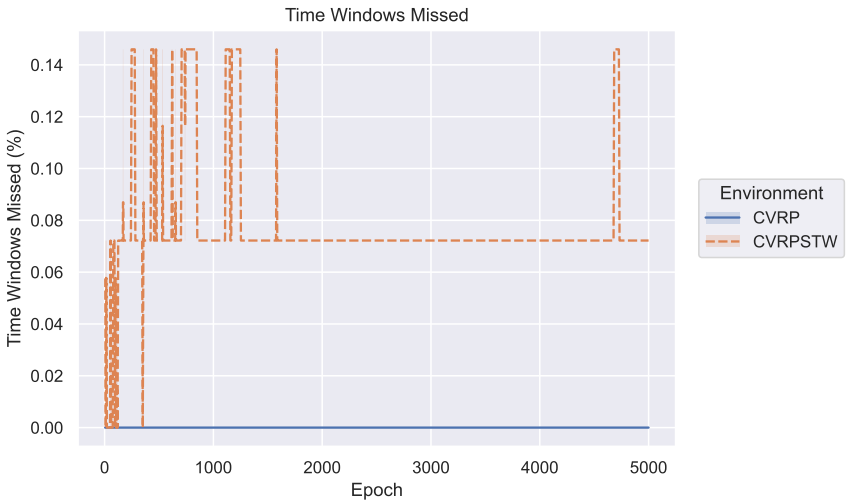


Figure 5.21: Time Windows Missed Comparison between Agents operating in Different Environments (Absolute Values)

As expected, the agent operating in the CVRP domain will start with better reward values as it will not be penalised by serving the clients outside the required time windows. However, we can note an unexpectedly significant reward difference between epochs 1000 and 2000. In fact, during that time, the *CVRPSTW* agent achieved a bigger reward and a smaller distance travelled, even though it was subject to more constraints and penalties than the *CVRP*.

Overall, although the agent operating in the CVRPSTW domain does not receive higher rewards, it still travels a somewhat similar distance. Additionally, its behaviour is more stable, and the number of time windows it misses is small. Only 0.14% of customers received their orders outside their expected time window, which means that from the 1.237% of customers that requested a time window, only 11.32% received theirs outside of it.

### 5.4.2 Node Features

Similarly to Section 5.3.2, this section evaluates the effects of different node features on the agent's performance. The most significant difference is related to the node attributes featured. The CVRP experiments' attributes were related to routing and capacity constraints. In contrast, in this experiment, they are related to time and time windows. Once again, we select some of the referred properties as obligatory and some as optional.

Table 5.6 identifies all the node features considered and whether they were obligatory or optional.

Table 5.6: Node Features Considered in the *CVRPSTW* Experiments

| **Node Feature** | **Obligatory** | **Optional** |
|---|---|---|
| *location* | X | |
| *depot* | X | |
| *demand* | X | |
| *current_capacity* | X | |
| *can_serve* | X | |
| *cost* | X | |
| *mask* | X | |
| *current_time* | X | |
| *travel_time* | X | |
| *service_time* | X | |
| *start_tw* | X | |
| *end_tw* | X | |
| *arrival_time* | | X |
| *time_after_service* | | X |
| *inside_tw* | | X |
| *tw_error* | | X |
| *after_tw_start* | | X |
| *before_tw_end* | | X |

The results of this experiment can be seen in Table 5.7. For simplicity, only the optional node features are presented in the table. The column *Accumulated Rewards* holds the cumulative rewards obtained through 25 training epochs. The best performant agents attain good accumulated rewards, as well as good initial and final performances. Additionally, the initial and final performances should be different to indicate the potential to develop and evolve over time. Figure 5.22 describes the distribution of the cumulative rewards obtained per agent.

Table 5.7: Optional Node Features Considered and Respective Accumulated Rewards in the *CVRPSTW* Environment

| Node Features | | | | | | Accumulated | Start | End |
|---|---|---|---|---|---|---|---|---|
| *arrival_time* | *time_after_service* | *inside_tw* | *tw_error* | *after_tw_start* | *before_tw_end* | Rewards | Reward | Reward |
| | | | | | | 12441.47 | 496.64 | 497.34 |
| X | | | | | | 12021.9 | 483.47 | 479.35 |
| | X | | | | | 12107.9 | 483.52 | 483.11 |
| | | X | | | | 11828.88 | 472.56 | 472.93 |
| | | | X | | | 11859.62 | 474.02 | 474.35 |
| | | | | X | | 11863.8 | 474.02 | 475.27 |
| | | | | | X | 11857.71 | 474.02 | 473.98 |
| X | X | | | | | 12401.76 | 496.81 | 495.8 |
| X | | X | | | | 12415.4 | 497.32 | 496.29 |
| X | | | X | | | 12437.28 | 497.44 | 497.35 |
| X | | | | X | | 12427.03 | 497.44 | 496.99 |
| X | | | | | X | 12424.52 | 497.44 | 496.54 |
| | X | X | | | | 12419.11 | 497.22 | 497.28 |
| | X | | X | | | 12412.12 | 497.34 | 496.47 |
| | X | | | X | | 12414.8 | 497.34 | 496.62 |
| | X | | | | X | 12408.41 | 497.34 | 496.41 |
| | | X | X | | | 12402.55 | 496.09 | 496.25 |
| | | X | | X | | 12396.79 | 496.09 | 496.05 |
| | | X | | | X | 12400.59 | 496.09 | 496.12 |
| | | | X | X | | 12403.86 | 496.25 | 496.19 |
| | | | X | | X | 12398.78 | 496.25 | 495.71 |
| | | | | X | X | 12395.9 | 496.25 | 495.6 |
| X | X | X | | | | 12109.65 | 481.55 | 489.5 |
| X | X | | X | | | 12108.67 | 481.51 | 489.85 |
| X | X | | | X | | 12163.3 | 481.51 | 490.59 |
| X | X | | | | X | 12110.01 | 481.51 | 489.84 |
| X | | X | X | | | 12123.91 | 485.95 | 484.6 |

Table 5.7: Optional Node Features Considered and Respective Accumulated Rewards in the *CVRPSTW* Environment

| Node Features | | | | | | Accumulated | Start | End |
|---|---|---|---|---|---|---|---|---|
| *arrival_time* | *time_after_service* | *inside_tw* | *tw_error* | *after_tw_start* | *before_tw_end* | Rewards | Reward | Reward |
| X | | X | | X | | 12145.92 | 485.95 | 488.55 |
| X | | X | | | X | 12184.01 | 485.95 | 487.46 |
| X | | | X | X | | 12148.46 | 486.0 | 485.84 |
| X | | | X | | X | 12169.04 | 486.0 | 489.2 |
| X | | | | X | X | 12134.6 | 486.0 | 483.25 |
| | X | X | X | | | 12147.89 | 485.78 | 486.21 |
| | X | X | | X | | 12115.91 | 485.78 | 483.08 |
| | X | X | | | X | 12190.97 | 485.78 | 488.92 |
| | X | | X | X | | 12173.0 | 485.73 | 490.22 |
| | X | | X | | X | 12129.14 | 485.73 | 484.85 |
| | X | | | X | X | 12163.33 | 485.73 | 483.16 |
| | | X | X | X | | 12068.64 | 483.3 | 482.74 |
| | | X | X | | X | 12111.3 | 483.3 | 485.08 |
| | | X | | X | X | 12102.9 | 483.3 | 486.79 |
| | | | X | X | X | 12079.34 | 483.19 | 476.57 |
| X | X | X | X | | | 11928.22 | 478.25 | 476.47 |
| X | X | X | | X | | 11940.47 | 478.25 | 477.72 |
| X | X | X | | | X | 11929.71 | 478.25 | 476.94 |
| X | X | | X | X | | 11943.19 | 477.73 | 478.46 |
| X | X | | X | | X | 11943.43 | 477.73 | 476.85 |
| X | X | | | X | X | 11921.56 | 477.73 | 476.1 |
| X | | X | X | X | | 11996.15 | 479.58 | 479.95 |
| X | | X | X | | X | 11984.06 | 479.58 | 479.25 |
| X | | X | | X | X | 12000.58 | 479.58 | 480.2 |
| X | | | X | X | X | 11921.26 | 476.84 | 477.26 |
| | X | X | X | X | | 12003.22 | 479.57 | 479.89 |
| | X | X | X | | X | 12018.8 | 479.57 | 481.27 |

Table 5.7: Optional Node Features Considered and Respective Accumulated Rewards in the *CVRPSTW* Environment

| Node Features | | | | | | Accumulated | Start | End |
|---|---|---|---|---|---|---|---|---|
| *arrival_time* | *time_after_service* | *inside_tw* | *tw_error* | *after_tw_start* | *before_tw_end* | Rewards | Reward | Reward |
| | X | X | | X | X | 12004.98 | 479.57 | 480.6 |
| | X | | X | X | X | 11925.72 | 476.73 | 476.61 |
| | | X | X | X | X | 11835.19 | 471.25 | 473.75 |
| X | X | X | X | X | | 12197.51 | 487.18 | 488.48 |
| X | X | X | X | | X | 12164.79 | 487.18 | 486.22 |
| X | X | X | | X | X | 12173.07 | 487.18 | 486.91 |
| X | X | | X | X | X | 12186.94 | 487.18 | 488.32 |
| X | | X | X | X | X | 12184.37 | 487.56 | 487.08 |
| | X | X | X | X | X | 12214.36 | 487.63 | 489.4 |
| X | X | X | X | X | X | 932.17 | 37.47 | 37.65 |

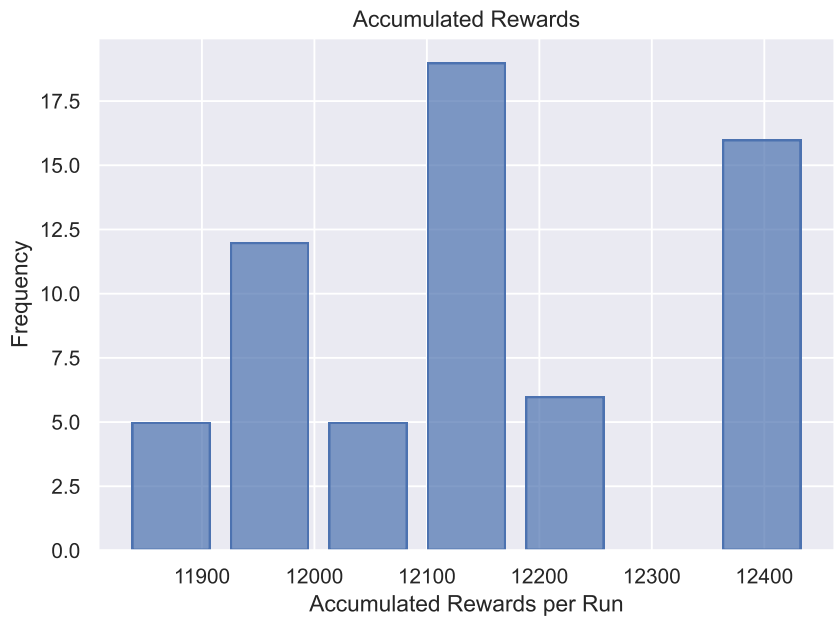


Figure 5.22: Distribution of Accumulated Rewards Obtained by Agents using Different Node Features through the Span of 25 Epochs

As can be seen, the variance in accumulated rewards is much smaller than the one obtained in Section 5.3.2. With the exception of using all optional node features, all other combinations perform much more similarly. However, there are still some disparities. Figure 5.23 showcases the effects of different node feature combinations throughout the 25 training epochs. It does not include the combo of all optional node features.

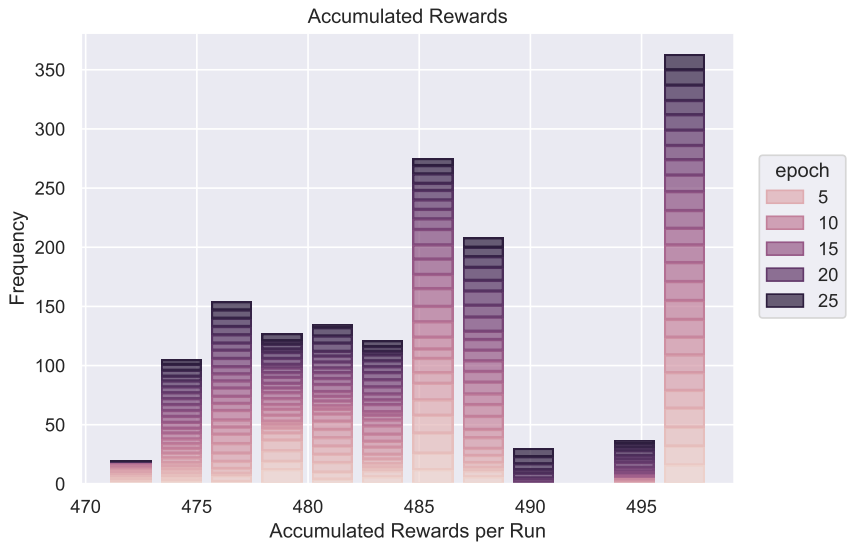


Figure 5.23: Distribution of Rewards per Epoch Obtained by Agents using Different Node Feature Combinations

It also observes a large range of values, with darker bars on the right, indicating that the node feature combinations allow the agent to learn and improve its performance. Nonetheless, with only 25 training epochs, the performance is still heavily affected by the initialisation method.

We decided to explore further the combination: *location*, *depot*, *demand*, *current_capacity*, *next_capacity*, *fuel_consumption*, *cost*, *current_time*, *travel_time*, *service_time*, *start_tw*, *end_tw*, *tw_error*, and *time_after_service* for the experiments regarding the *CVRP* environment.

### 5.4.3 Reward Functions

This section assumes a similar purpose to Section 5.3.3: to explore the impact and performance of different reward functions. However, instead of testing the reward signals related to routing and the capacity constraints, we explore the ones related to the customer time windows. Once again, due to the nature of this section and to allow a fair comparison between agents, the absolute rewards per episode will not be considered.

Figures 5.24, 5.25 and 5.26 portray the evolution of agents operating in the *CVRPSTW* environment.

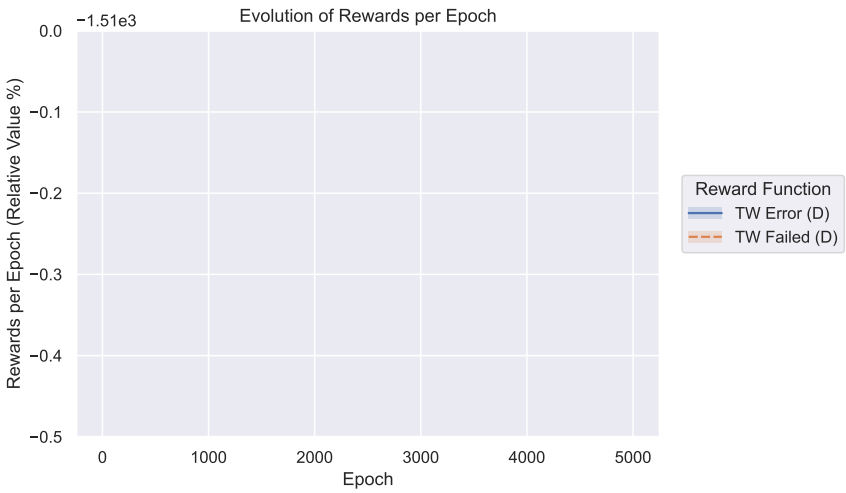


Figure 5.24: Rewards per Episode Comparison between Agents using Different Reward Functions (Relative Values)

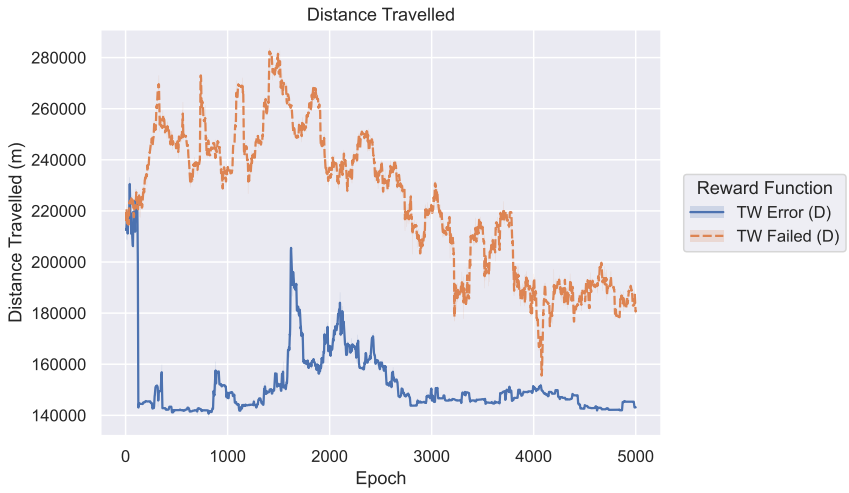


Figure 5.25: Distance Travelled Comparison between Agents using Different Reward Functions (Absolute Values)

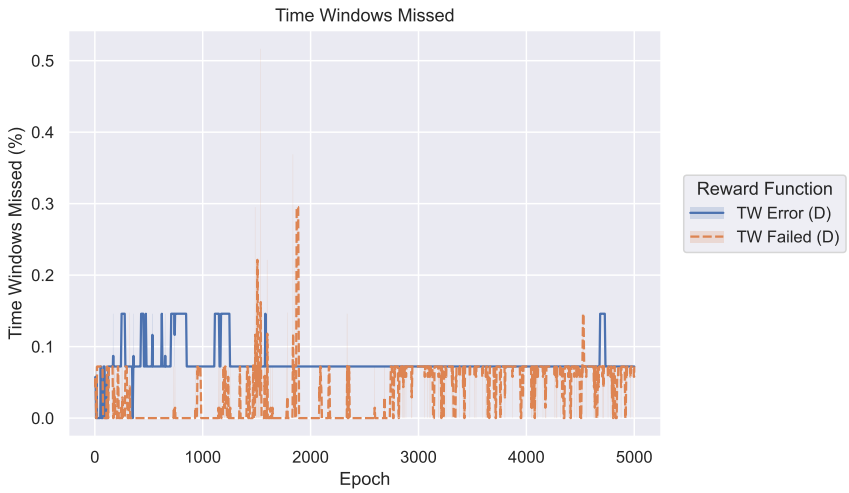


Figure 5.26: Failed Time Windows Comparison between Agents using Different Reward Functions (Absolute Values)

Figure 5.27 demonstrates how the penalty applied evolves as the delivery gets farther from the requested time window. In this figure, the *Error Reward* is the penalty applied to the *TW Error (D)*, and the *Inside or Outside Reward* is the penalty applied to the *TW Failed (D)*. The values correspond to penalties, which, essentially, means that the agents receive them as negative values, the symmetric of the value represented in the graph.

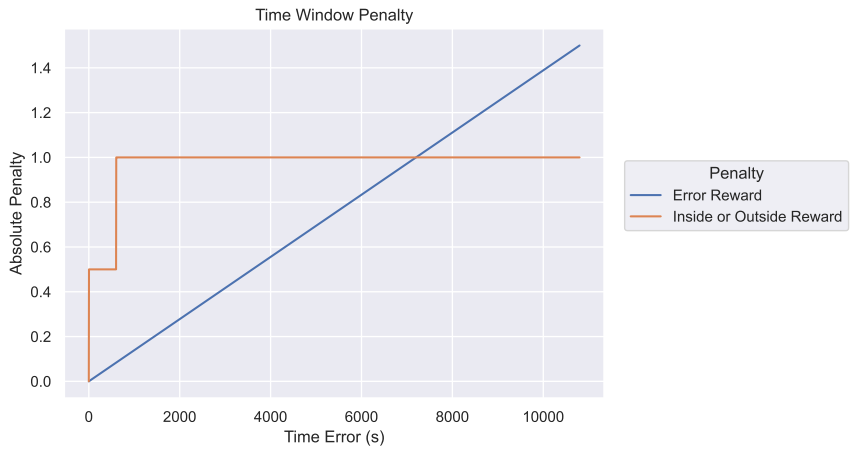


Figure 5.27: Penalties Obtained as the Delivery Time gets Farther Away from the Requested Time Window

Since the *REINFORCE* algorithm is based on gradients, a constant function during a certain portion of values is harder to optimise. Due to the nature of the reward functions, and the penalty applied to the agents for missing deliveries on the requested time windows (shown in Figure 5.27), the agent receiving the *TW Error (D)* will converge quicker than the agent receiving the *TW Failed (D)* reward.

## 5.5 Exploring Transfer Learning

This section explores the effects of transferring the knowledge from a source model to a target model and continuing its training on the new task. To transfer that knowledge from one model to another, we simply initialise the weights and biases of the target model with the respective values held by the source model. This practice is called fine-tuning, one of the Transfer Learning approaches. We chose to fine-tune our models without freezing any layers (frozen layers do not update during the backpropagation step). By doing this, our entire network is allowed to train and adapt to the new task.

### 5.5.1 Changing Dataset

The first experiment regarding Transfer Learning explores the ability of the agent to adapt to a different dataset. Firstly, the agent is trained on the Amazon Dataset, considering a CVRP environment. Afterwards, the knowledge is transferred to another agent operating on the Loggi Dataset, still in a CVRP environment. Below, we compare the performance of two agents, one that received knowledge from the previously trained agent and another that started the training from scratch.

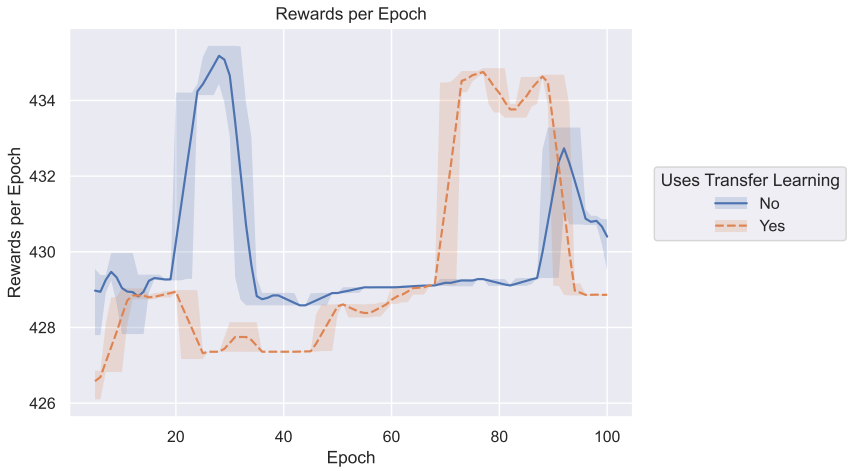


Figure 5.28: Rewards per Episode Comparison between Agents when using Transfer Learning to Change Dataset (Absolute Values)

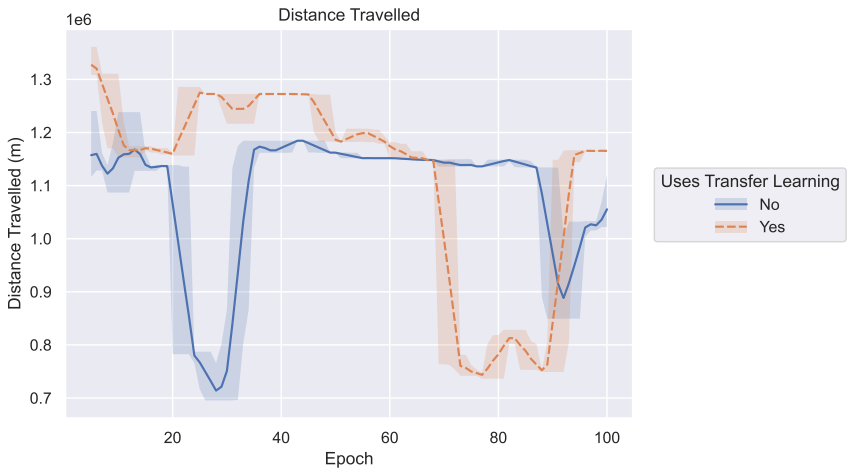


Figure 5.29: Distance Travelled Comparison between Agents when using Transfer Learning to Change Dataset (Absolute Values)

The similarities between both datasets affect how positive the transfer is. Both the source and target agents operate in the same environment, and only the magnitude of values changes. Since the domains are very similar, the agent performs much better than the one trained from scratch. Even though the agent that starts the training from scratch has a similar start to the one subject to knowledge transfer, the latter quickly surpasses its counterpart. It becomes more stable and receives more rewards.

### 5.5.2 Removing the Mask

Our second experiment involving Transfer Learning evaluates the ability of the agent to function without the mask. Figure 5.30 illustrates the effects of reusing knowledge from an agent trained with it.

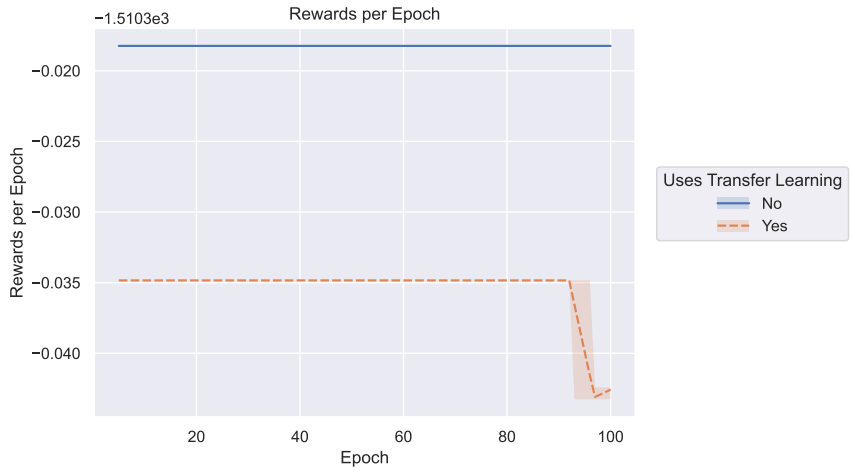


Figure 5.30: Rewards per Episode Comparison between Agents when Using Transfer Learning to Remove the Mask (Absolute Values)

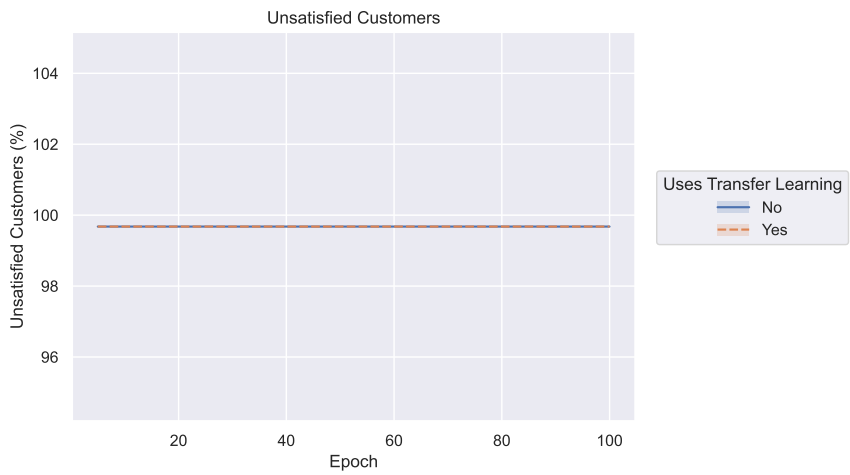


Figure 5.31: Unsatisfied Customers Percentage Comparison between Agents when Using Transfer Learning to Remove the Mask (Absolute Values)

Since the mask proved such a critical piece in the agent's training, it was expectable that the agent subject to knowledge transfer surpassed the agent without TL substantially. However, the agent is not capable of outperforming its counterpart. It starts by scoring $-1510.33$, only marginally below (0.02 units) the agent without knowledge transfer, who scored $-1510.31$. Then, it continues to perform slightly worse than the agent without Transfer Learning. However, it

evolves quicker as its first downward curve appears before the 100 epochs, while its counterpart only starts evolving around the 800 epochs mark, as seen in Section 5.3.1.

There are two main possibilities for this low performance: the lack of training of the source agent and the exploring phase of RL. On the one hand, the source agent might require more training before being generalisable to working without the mask. On the other hand, since the source agent never picked invalid nodes, it might not recognise them as invalid but rather unexplored. It would require longer training to discern which option applies or if some other factors are affecting the target agent.

### 5.5.3 Changing Environment

The third experiment explores one of the focal topics of this project, whether or not knowledge can be reused across different VRP variants. We start by analysing how an agent trained in the CVRPSTW environment performs in the CVRP environment (in Figures 5.32, 5.33 and 5.34).

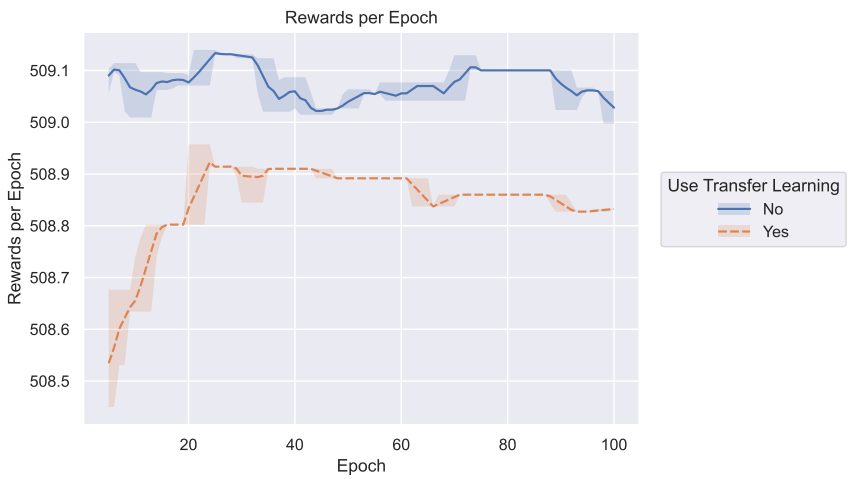


Figure 5.32: Rewards per Episode Comparison between Agents when using Transfer Learning to Change to an Easier Environment (Absolute Values)

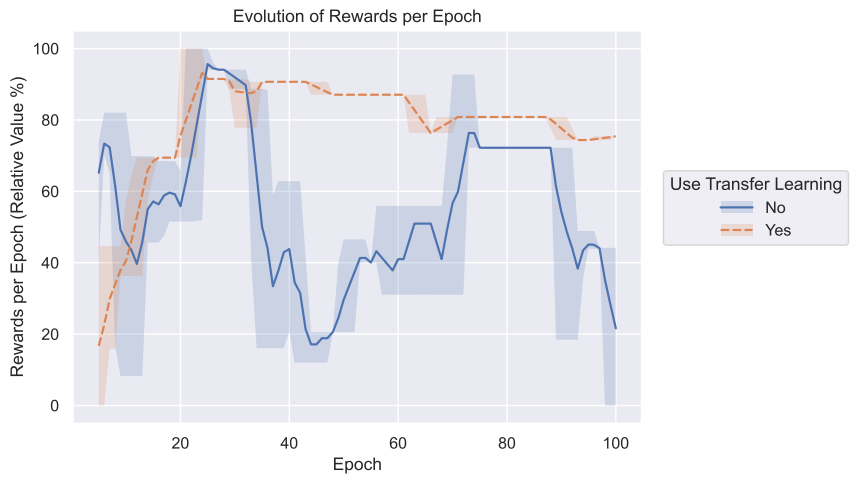


Figure 5.33: Rewards per Episode Comparison between Agents when using Transfer Learning to Change to an Easier Environment (Relative Values)

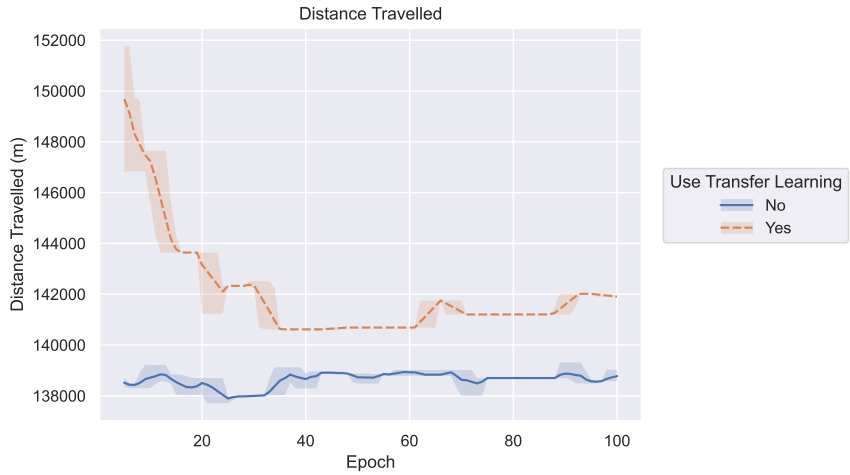


Figure 5.34: Distance Travelled Comparison between Agents when using Transfer Learning to Change to an Easier Environment (Absolute Values)

Since the target task involves fewer constraints than those in which the agent was originally trained (in the source task), the agent would be expected to perform exceptionally well. However, contrary to expectations, this does not happen. The agent subject to Transfer Learning improves faster than its counterpart. However, even with those improvements, it still underperforms. In fact, it only receives 98.72% of the total rewards received by the agent starting from scratch, receiving a reward of 50254.86 over 100 epochs, while its counterpart receives 50907.42.

However, when an agent trained in the CVRP environment is placed in a CVRPSTW environment, it thrives (Figure 5.37). The distance travelled is much smaller in the agent subject to knowledge transfer, as can be observed in Figure 5.35.

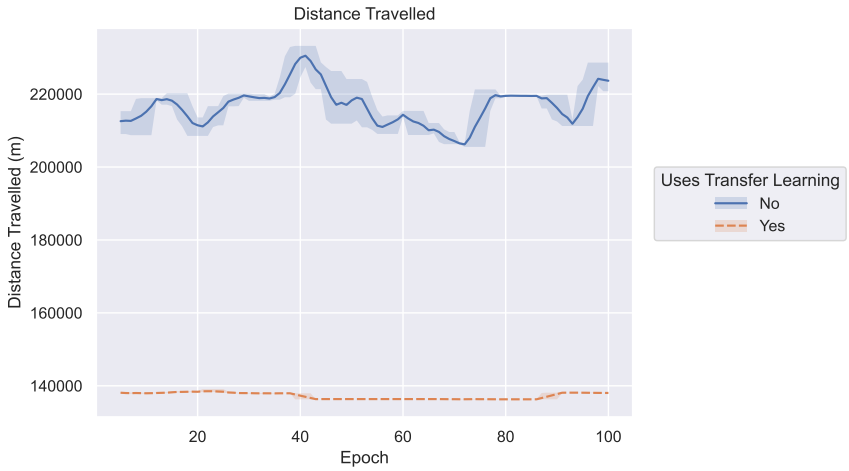


Figure 5.35: Distance Travelled Comparison between Agents when using Transfer Learning to Change to an Harder Environment (Absolute Values)

Additionally, the agent also manages to serve more customers within their respective time windows, shown in Figure 5.36, leading to an even higher difference in the received rewards.

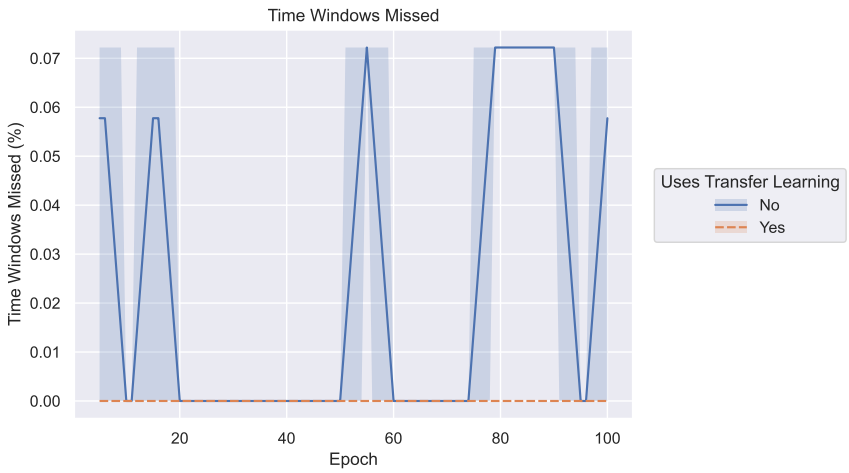


Figure 5.36: Time Windows Missed Comparison between Agents when using Transfer Learning to Change to an Harder Environment (Absolute Values)

Figure 5.37 depicts the difference in the rewards. This time, the agent subject to Transfer Learning outperforms its counterpart. Indeed, the agent *not* subject to Transfer Learning collects 98.89% of the rewards received by the agent who undergoes knowledge transfer. It only manages a total reward of 50328.32, while its counterpart outperforms it, collecting 50893.78 over the 100 epochs.

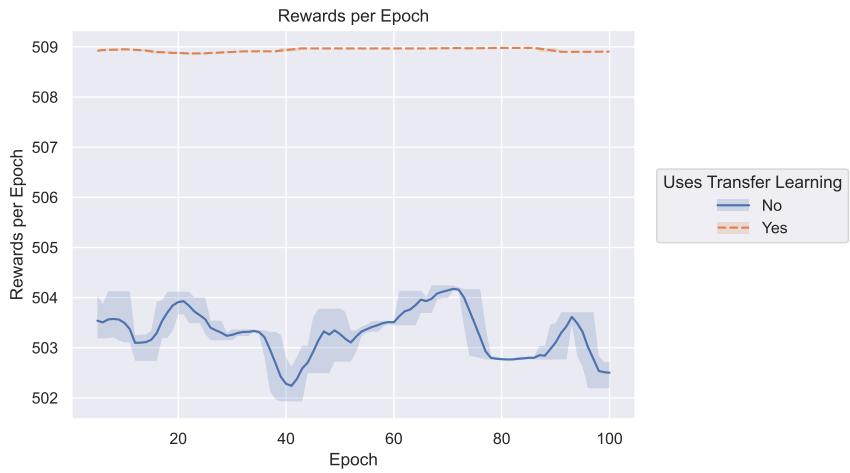


Figure 5.37: Rewards per Episode Comparison between Agents when using Transfer Learning to Change to an Harder Environment (Absolute Values)

These contradictory findings, where the knowledge is transferable when the problem gets harder but not when it gets easier, reinforce the importance of understanding the effects of Transfer Learning. One possible reason for these inconsistencies is the lack of training undergone by the source agent. Since the agent did not reach an optimal performance, the quality of the knowledge is not optimal. As such, the target agent underperforms when compared to learning from scratch.

In fact, Section 5.4.1 compared the learning and evolution of two agents acting in distinct environments: *CVRP* and *CVRPSTW*. At first, this comparison seems absurd, as the agents function in different environments under different reward functions. However, it helps us understand why transferring the knowledge between the CVRPSTW and the CVRP leads to such unexpectedly poor performance. When we look at those graphs, we can note the lack of quality of the *CVRP* agent, which at its peak, collects only 0.54 more rewards than the *CVRPSTW* (standing at 509.42 against 508.88).

### 5.5.4 Changing Reward Functions

After demonstrating the effects of reward functions on the agent's learning and performance, it is time to evaluate whether an agent trained on a reward function can be generalised to actuate with another. For that, we devised two experiments, one in the CVRP domain and another in the CVRPSTW domain.

Firstly, we start with the CVRP domain. Figures 5.38, 5.39, 5.40, 5.41, 5.42, and 5.43 compare the performance of the agent with and without Transfer Learning when learning a new reward function in the CVRP environment.

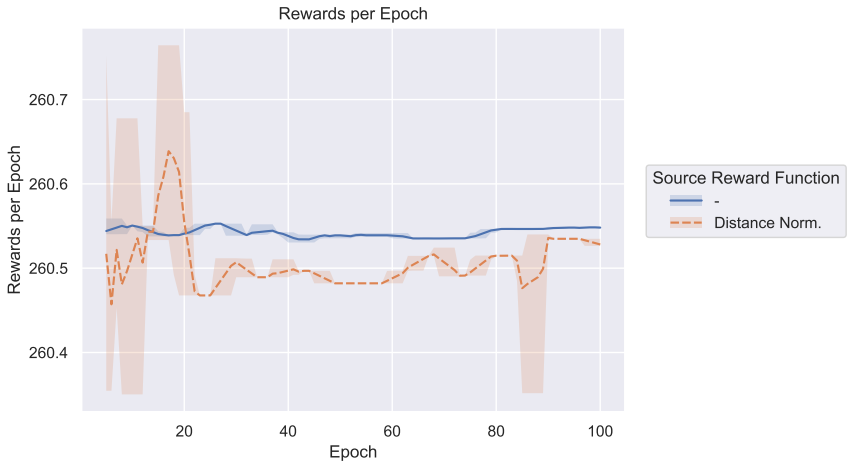


Figure 5.38: Rewards per Episode Comparison between Agents when using Transfer Learning to Change Reward Function to *Fraction* (Absolute Values)

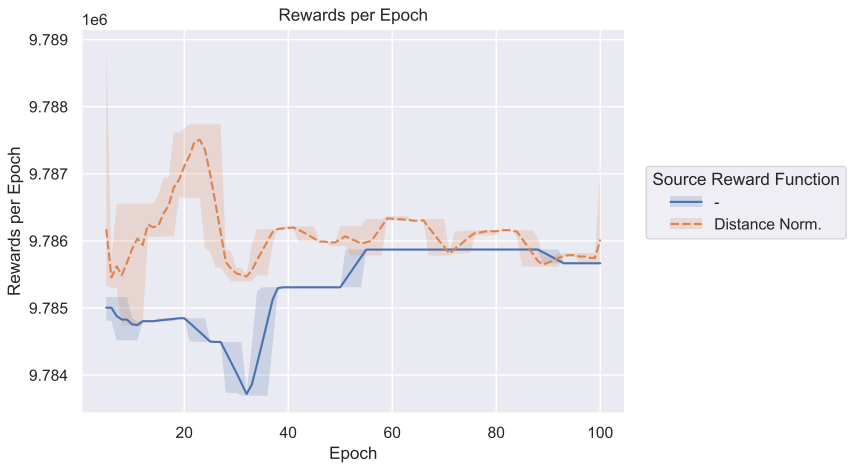


Figure 5.39: Rewards per Episode Comparison between Agents when using Transfer Learning to Change Reward Function to *Savings* (Absolute Values)

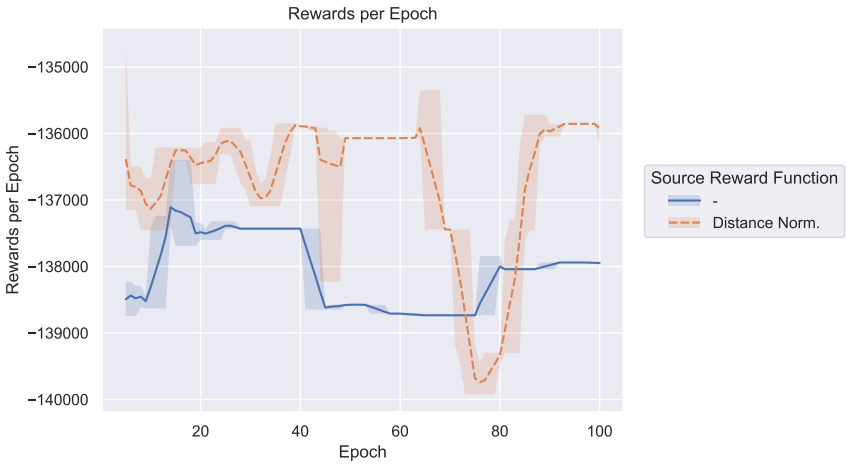


Figure 5.40: Rewards per Episode Comparison between Agents when using Transfer Learning to Change Reward Function to *Distance* (Absolute Values)

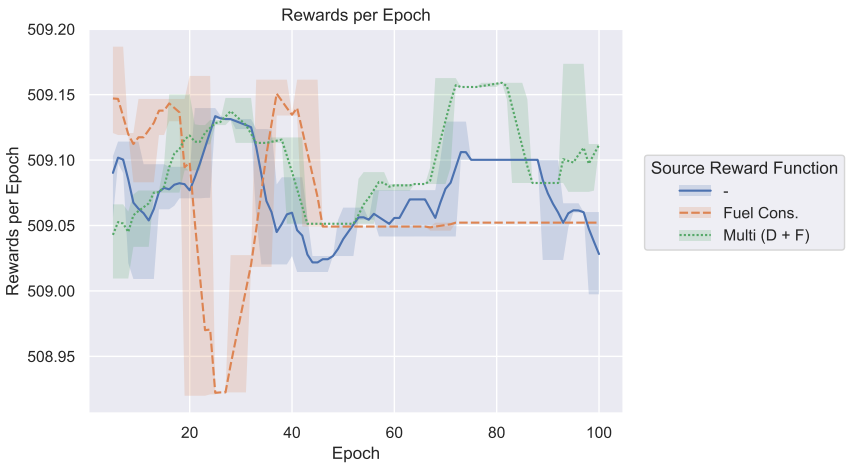


Figure 5.41: Rewards per Episode Comparison between Agents when using Transfer Learning to Change Reward Function to *Distance Norm.* (Absolute Values)

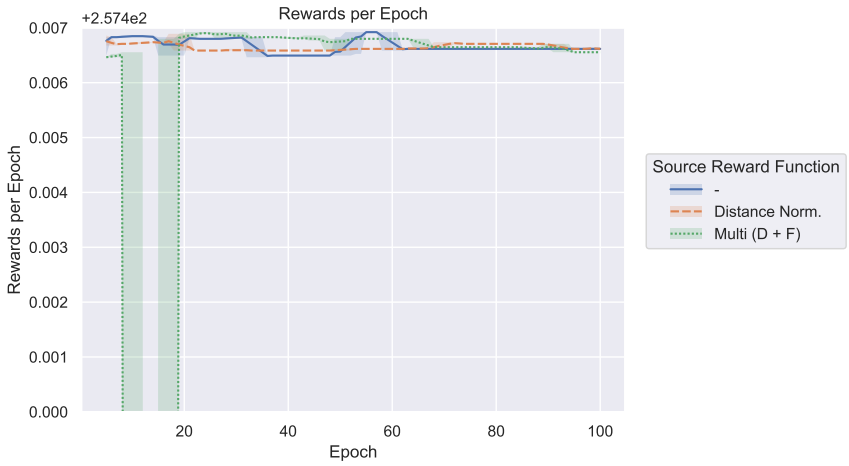


Figure 5.42: Rewards per Episode Comparison between Agents when using Transfer Learning to Change Reward Function to *Fuel Cons.* (Absolute Values)

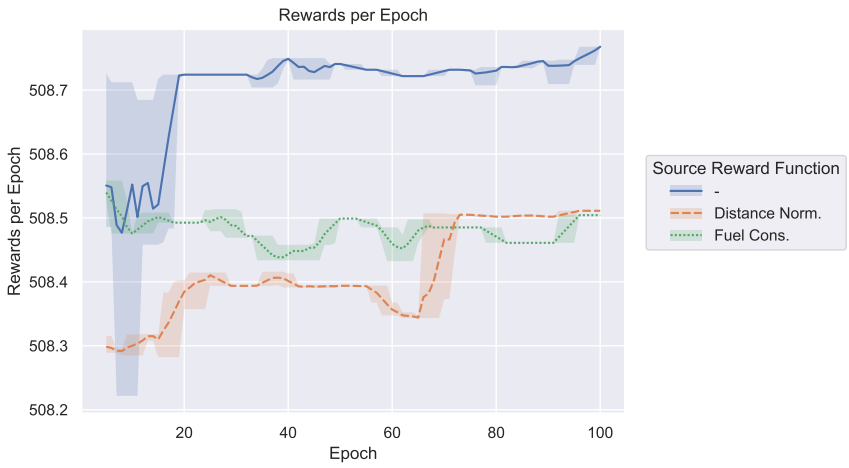


Figure 5.43: Rewards per Episode Comparison between Agents when using Transfer Learning to Change Reward Function to *Multi (D + F)* (Absolute Values)

As expected, the most favourable transfer occurs when changing between reward functions that consider the same objective. For example, the *Distance*, *Distance Norm.*, *Fraction*, and *Savings* rewards attempt to guide the agent to minimise the total distance travelled. Therefore, even if the absolute value of the received rewards changes, the optimal behaviour does not. Indeed, the optimal action in the source domain is still the optimal action in the target domain. Additionally, an action that received a smaller reward in the source task would still get awarded less in the target domain. Nonetheless, the *Fraction* reward function seems to behave differently, and the agent learning from scratch in it is quite stable and able to outperform its counterpart subject to knowledge transfer.

On the other hand, when attempting to change between reward functions that aim to optimise different objectives, such as the *Fuel Cons.* and the *Distance Norm.*, the benefits are not as easily attained. When transferring knowledge to operate with the *Distance Norm.* reward function, it is not clear which agent performs better. Indeed, their performances are really close, with the agent starting from scratch collecting a total reward of 509.14 over the 100 epochs, while the agents who transferred knowledge from *Fuel Cons.* and *Multi (D + F)* received 509.19 and 509.17, respectively. When transferring knowledge to operate with the *Fuel Cons.* reward function, all the agents perform almost equally, and the agent starting from scratch with the *Multi (D + F)* reward function outperforms its counterparts subject to TL.

Then, we experiment in the CVRPSTW domain. Here, we compare the agent's performance when learning a reward function with a different penalty for missing time windows in Figures 5.44 and 5.45.

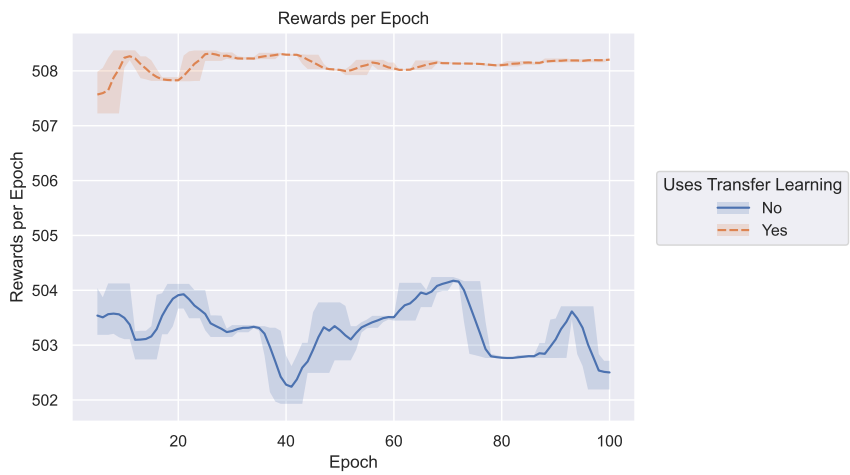


Figure 5.44: Rewards per Episode Comparison between Agents when using Transfer Learning to Change Reward Function to *TW Error (D)* (Absolute Values)

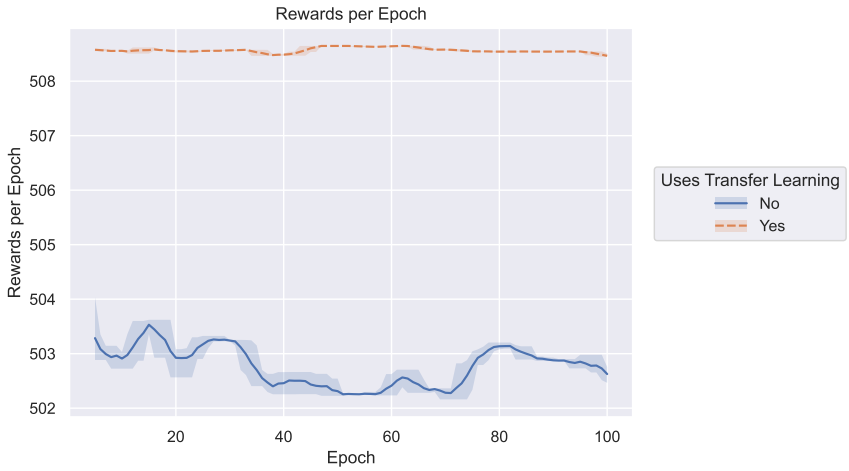


Figure 5.45: Rewards per Episode Comparison between Agents when using Transfer Learning to Change Reward Function to *TW Failed (D)* (Absolute Values)

As seen before, the relationship between reward functions influences the transfer quality. Figure 5.46 demonstrates how the penalty applied evolves as the delivery gets farther from the requested time window. In this figure, the *Error Reward* is the penalty applied to the *TW Error (D)*, and the *Inside or Outside Reward* is the penalty applied to the *TW Failed (D)*. The values correspond to penalties, which, essentially, means that the agents receive them as negative values, the symmetric of the value represented in the graph.

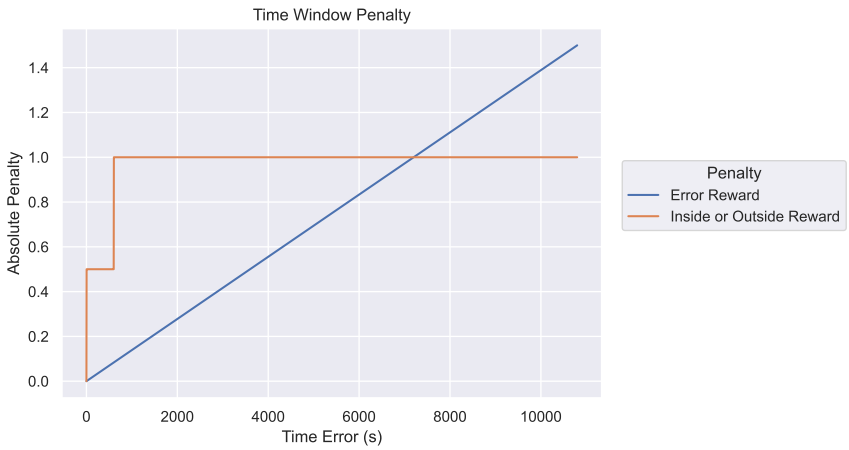


Figure 5.46: Penalties Obtained as the Delivery Time gets Farther Away from the Requested Time Window

In this case, there is some relation between reward functions, as seen in Figure 5.46. However, actions with the same penalty on the *TW Failed (D)* reward function might have different penalties in the *TW Error (D)* reward function. As such, the connection is not as easy as the one witnessed in

the *CVRP* between the reward functions attempting to minimise the distance travelled. Nonetheless, the performance still transfers well to the new agent achieving performances of 0.82% and 0.91% above the agents starting from scratch when transferring to *TW Error (D)* and *TW Failed (D)*, respectively.

# Chapter 6

# Conclusions

With the proliferation of e-commerce, there has been an increasing demand for cheap and fast deliveries. Besides, more than ever before, environmental issues and ecological concerns are growing wider. These occurrences make a system capable of responding quickly, considering multiple optimisation objectives, and adapting to new and different circumstances, a very desirable tool. Reinforcement Learning (RL) is an effective methodology already applied in many fields, including the transportation and delivery sectors. Although its use in real-world scenarios faces multiple challenges, there is already work attempting to solve some of them. Transfer Learning (TL) is one of the techniques used to mitigate some of the problems RL agents face.

The existing literature in the field portrays representation as a vital problem both for RL and TL. It points Graph Neural Networks as a potential tool to solve Vehicle Routing Problems with a variable number of customers due to their capacity to deal with variable-sized graphs. Additionally, it highlights the potential of TL to improve sample efficiency and reduce the number of samples necessary to train an RL agent.

In this project, we explored the application of Deep Reinforcement Learning (DRL) with Graph Convolutional Networks to solve two variants of the Vehicle Routing Problem: the Capacitated Vehicle Routing Problem and the Capacitated Vehicle Routing Problem with Soft Time Windows. Multiple experiments were devised to compare the RL agent's performance when operating under diverse reward functions and receiving different information from the environment. Moreover, TL was used to attempt to achieve a performant model with reduced training.

Our main findings can be mainly divided into three groups. In a combinatorial problem, limiting the search space with a masking scheme significantly increases the model performance. Besides, all our results point to the importance of a correct initialisation of the model and the correct shaping of a reward function. Finally, although it depends on the source model's quality, we proved that Transfer Learning can be successful when changing VRP variants or reward functions.

## 6.1 Main Contributions

This work aims to contribute not only to solving routing problems in smart cities but primarily to the field of Reinforcement Learning and the application of Transfer Learning on Reinforcement Learning agents.

Through an extensive literature review on how to solve Vehicle Routing Problems (VRP) with the application of DRL and Graph Deep Reinforcement Learning (GDRL), we identified the components and dimensions of the VRP, as well as the current efforts and techniques used to solve it with RL. Additionally, we were able to define a taxonomy for the VRP solved using RL, which can be used to systematise knowledge of the problem.

From the representation perspective, besides adopting Deep Reinforcement Learning as an optimisation approach to the VRP, this dissertation resorts to Graph Neural Networks, making explicit use of the nature of the problem, which is network-based. Moreover, facing the curse of dimensionality, which ultimately yields long, unnecessary exploration phases of invalid actions, this work suggests a practical approach to addressing these issues by designing appropriate masking techniques.

We devised an exploratory empirical study on the applicability and effects of using Transfer Learning to leverage training efforts among similar VRP instances. Although preliminary, it shows promising gains when Transfer Learning is employed to VRP variants and change reward functions.

Finally, it is important to notice the pipeline devised to handle the different datasets used in this work, which contributed to demonstrating the approach proposed in this work in real-world and large-scale delivery networks.

## 6.2 Future Work

As for future work, we propose two different avenues: improvements to the current project and new projects that extend or derive from it.

Firstly, when it comes to improvements, we could start by further optimising the hyperparameters. Additionally, we could explore the possibility of using different hyperparameters when dealing with different reward functions. Other architectures, a different layer organisation, and different RL algorithms, such as A2C or PPO, could also be tested. In addition, other TL techniques could be considered.

In order to enrich the analytical part of our work, the computational setup could be modified to allow for more efficient and extended training periods. This would enable us to run the experiments for more epochs and further explore the effects of Transfer Learning. Moreover, we could first experiment with training on artificial data and then transfer the knowledge to an agent operating

in real-world datasets. On top of all that, we could implement graph sparsification techniques to allow solving of even larger instances and other masking mechanisms.

As an extension to the current work, a couple of steps still need to be taken for this model to be implementable in a real-world scenario. First of all, we could apply the proposed approach in Multi-Agent RL (MARL) settings, exploring distributed and decentralised solutions. In addition, other concepts and concerns in AI, such as explainability, transparency or fairness, could also be explored to leverage actionable decision-making.

Finally, other VRP variants, including new real-world constraints, might need to be contemplated. Moreover, dynamically calculating the shortest paths between delivery locations allows for deviations in the distance, time taken, and fuel consumption according to the current traffic conditions. The final step would be deploying the proposed methodological approach within a digital twin of transport planning frameworks to support management operations in urban logistics.

# References


[1] Alekh Agarwal, Yuda Song, Wen Sun, Kaiwen Wang, Mengdi Wang, and Xuezhou Zhang. Provable benefits of representational transfer in reinforcement learning, 2022.

[2] Lucas Nunes Alegre, Ana Bazzan, and Bruno C. Da Silva. Optimistic linear support and successor features as a basis for optimal policy transfer. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 394–413. PMLR, 17–23 Jul 2022.

[3] Majed G. Alharbi, Ahmed Stohy, Mohammed Elhenawy, Mahmoud Masoud, and Hamiden Abd El-Wahed Khalifa. Solving pickup and drop-off problem using hybrid pointer networks with deep reinforcement learning. *PLOS ONE*, 17(5):1–19, 05 2022.

[4] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay, 2017.

[5] Leo Ardon. Reinforcement learning to solve np-hard problems: an application to the cvrp, 2022.

[6] Mohammad Asghari and S. Mohammad J. Mirzapour Al-e-hashem. Green vehicle routing problem: A state-of-the-art review. *International Journal of Production Economics*, 231:107899, 2021.

[7] Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[8] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning, 2017.

[9] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[10] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning, 2016.

[11] Matteo Boffa, Zied Ben Houidi, Jonatan Krolikowski, and Dario Rossi. Neural combinatorial optimization beyond the tsp: Existing architectures under-represent graph structure, 2022.

[12] Yuhong Cao, Zhanhong Sun, and Guillaume Sartoretti. Dan: Decentralized attention-based neural network for the minmax multiple traveling salesman problem, 2022.

[13] Jinwei Chen, Zefang Zong, Yunlin Zhuang, Huan Yan, Depeng Jin, and Yong Li. Reinforcement learning for practical express systems with mixed deliveries and pickups. *ACM Trans. Knowl. Discov. Data*, 17(3), feb 2023.

[14] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[15] John R. Current and David A. Schilling. The covering salesman problem. *Transportation Science*, 23(3):208–213, 1989.

[16] Przemysław Czuba and Dariusz Pierzchała. Machine learning methods for solving vehicle routing problems. In *36th International Business Information Management Association (IBIMA)*, 01 2021.

[17] Will Dabney, Mark Rowland, Marc G. Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression, 2017.

[18] Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs, 2017.

[19] Oscar Day and Taghi M. Khoshgoftaar. A survey on heterogeneous transfer learning. *Journal of Big Data*, 4(1):29, Sep 2017.

[20] Aymar de Brouchoven de Bergeyck. An iterative deep reinforcement learning approach to solve vehicle routing problems. Master's thesis, Delft University of Technology, 10 2022. http://resolver.tudelft.nl/uuid:09b527d1-de01-4a40-941c-7ba6f9368423.

[21] Google Developers. Representation | machine learning | google developers. https://developers.google.com/machine-learning/crash-course/representation/video-lecture, 07 2022. [Online; accessed 5-February-2023].

[22] Google Developers. Vehicle Routing | OR-Tools | Google Developers. https://developers.google.com/optimization/routing, 01 2023. [Online; accessed 5-February-2023].

[23] Shelagh Dolan. Last mile delivery logistics explained: Problems & solutions. https://www.insiderintelligence.com/insights/last-mile-delivery-shipping-explained/, Jan 2023. [Online; accessed 5-February-2023].

[24] Marco Dorigo. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano*, 1992.

[25] Gabriel Dulac-Arnold, Nir Levine, Daniel J. Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, Sep 2021.

[26] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning, 2019.

[27] Mhairi Dunion, Trevor McInroe, Kevin Sebastian Luck, Josiah Hanna, and Stefano V. Albrecht. Temporal disentanglement of representations for improved generalisation in reinforcement learning, 2022.

[28] Nahid Parvez Farazi, Tanvir Ahamed, Limon Barua, and Bo Zou. Deep reinforcement learning and transportation research: A comprehensive review, 2020.

[29] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I. Jordan, Joseph E. Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning, 2018.

[30] Getu Fellek, Ahmed Farid, Goytom Gebreyesus, Shigeru Fujimura, and Osamu Yoshie. Graph transformer with reinforcement learning for vehicle routing problem. *IEEJ Transactions on Electrical and Electronic Engineering*, 18(5):701–713, 2023.

[31] Liang Feng, Yuxiao Huang, Lei Zhou, Jinghui Zhong, Abhishek Gupta, Ke Tang, and Kay Chen Tan. Explicit evolutionary multitasking for combinatorial optimization: A case study on capacitated vehicle routing problem. *IEEE Transactions on Cybernetics*, 51(6):3143–3156, 2021.

[32] Júlio César Ferreira, Maria Teresinha Arns Steiner, and Osíris Canciglieri Junior. Multi-objective optimization for the green vehicle routing problem: A systematic literature review and future directions. *Cogent Engineering*, 7(1):1807082, 2020.

[33] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[34] Simone Foa, Corrado Coppola, Giorgio Grani, and Laura Palagi. Solving the vehicle routing problem with deep reinforcement learning, 2022.

[35] OpenStreetMap Foundation. Main page — openstreetmap foundation,, 2023. [Online; accessed 4-July-2023].

[36] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.

[37] Chenchen Fu, Zhengxuan Gao, Weiwei Wu, Vincent Chau, Jie Wang, Xueyong Xu, and Junzhou Luo. A learning approach for multi-agent travelling problem with dynamic service requirement in mobile iot. *Computers and Electrical Engineering*, 104:108397, 2022.

[38] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

[39] Daniele Gammelli, Kaidi Yang, James Harrison, Filipe Rodrigues, Francisco C. Pereira, and Marco Pavone. Graph neural network reinforcement learning for autonomous mobility-on-demand systems, 2021.

[40] Fred Glover. Future paths for integer programming and links to artificial intelligence." computers & operations research 13, 533-549. *Computers & Operations Research*, 13:533–549, 01 1986.

[41] The World Bank Group. Urban development overview. https://www.worldbank.org/en/topic/urbandevelopment/overview, Oct 2022. [Online; accessed 5-February-2023].

[42] Abhinav Gupta, Supratim Ghosh, and Anulekha Dhara. Deep reinforcement learning algorithm for fast solutions to vehicle routing problem with time-windows. In *5th Joint International Conference on Data Science & Management of Data (9th ACM IKDD CODS and 27th COMAD)*, CODS-COMAD 2022, page 236–240, New York, NY, USA, 2022. Association for Computing Machinery.

[43] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems 31*, pages 2451–2463. Curran Associates, Inc., 2018. https://worldmodels.github.io.

[44] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[45] Joyce Hendriks. Towards applying reinforcement learning to the home care scheduling and routing problem. Master's thesis, Tilburg School of Economics and Management - Tilburg University, 2022. http://arno.uvt.nl/show.cgi?fid=158619.

[46] André Hottung, Yeong-Dae Kwon, and Kevin Tierney. Efficient active search for combinatorial optimization problems, 2022.

[47] 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*, 2023.

[48] Shengyi Huang and Santiago Ontañón. A closer look at invalid action masking in policy gradient algorithms. *The International FLAIRS Conference Proceedings*, 35, may 2022.

[49] David Jonassen. Using cognitive tools to represent problems. *Journal of Research on Technology in Education*, 35(3):362–381, 2003.

[50] Chaitanya K. Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning TSP Requires Rethinking Generalization. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, volume 210 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:21, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[51] Minsu Kim, Jinkyoo Park, and Joungho Kim. Learning collaborative policies to solve np-hard routing problems, 2021.

[52] Minsu Kim, Jinkyoo Park, and joungho kim. Learning collaborative policies to solve np-hard routing problems. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 10418–10430. Curran Associates, Inc., 2021.

[53] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[54] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[55] W. Kool. *Learning and optimization in combinatorial spaces*. PhD thesis, Faculty of Science (FNWI), Amsterdam, 2022. https://hdl.handle.net/11245.1/b8d1289e-8204-49ba-b3c1-d11dbb613ef1.

[56] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems!, 2018.

[57] Nicholas D. Kullman, Jorge E. Mendoza, Martin Cousineau, and Justin C. Goodson. Atarifying the vehicle routing problem with stochastic service requests, 2019.

[58] Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. Matrix encoding networks for neural combinatorial optimization. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 5138–5149. Curran Associates, Inc., 2021.

[59] Annu Lambora, Kunal Gupta, and Kriti Chopra. Genetic algorithm- a literature review. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 380–384, 2019.

[60] Charline Le Lan, Stephen Tu, Adam Oberman, Rishabh Agarwal, and Marc G. Bellemare. On the generalization of representations in reinforcement learning, 2022.

[61] Jaeho Lee and Young Jae Jang. The graph neural network-based dynamic routing algorithm for overhead hoist transport vehicles in semiconductor fabrication plants. In *Proceedings of the 2022 International Symposium on Semiconductor Manufacturing Intelligence (ISMI2022)*, Tokyo, Japan, 2022. Proceedings of the 2022 International Symposium on Semiconductor Manufacturing Intelligence (ISMI2022). https://ciie2022.conf.tw/site/userdata/1449/ISMI_paper/ISMI2022_paper_5130.pdf.

[62] Kun Lei, Peng Guo, Yi Wang, Xiao Wu, and Wenchao Zhao. Solve routing problems with a residual edge-graph attention neural network. *Neurocomputing*, 508:79–98, 2022.

[63] Jingwen Li, Yining Ma, Ruize Gao, Zhiguang Cao, Andrew Lim, Wen Song, and Jie Zhang. Deep reinforcement learning for solving the heterogeneous capacitated vehicle routing problem. *IEEE Transactions on Cybernetics*, 52(12):13572–13585, 2022.

[64] Jingwen Li, Liang Xin, Zhiguang Cao, Andrew Lim, Wen Song, and Jie Zhang. Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 23(3):2306–2315, 2022.

[65] Kaiwen Li, Tao Zhang, Rui Wang, Yuheng Wang, Yi Han, and Ling Wang. Deep reinforcement learning for combinatorial optimization: Covering salesman problems. *IEEE Transactions on Cybernetics*, 52(12):13142–13155, 2022.

[66] 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, 2015.

[67] Bo Lin, Bissan Ghaddar, and Jatin Nathwani. Deep reinforcement learning for the electric vehicle routing problem with time windows. *IEEE Transactions on Intelligent Transportation Systems*, 23(8):11528–11538, 2022.

[68] Jane Lin, Wei Zhou, and Ouri Wolfson. Electric vehicle routing problem. *Transportation Research Procedia*, 12:508–521, 2016. Tenth International Conference on City Logistics 17-19 June 2015, Tenerife, Spain.

[69] Xi Lin, Zhiyuan Yang, and Qingfu Zhang. Pareto set learning for neural multi-objective combinatorial optimization, 2022.

[70] Zhengxuan Ling, Yu Zhang, and Xi Chen. A deep reinforcement learning based real-time solution policy for the traveling salesman problem. *IEEE Transactions on Intelligent Transportation Systems*, 24(6):5871–5882, 2023.

[71] Fei Liu, Chengyu Lu, Lin Gui, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. Heuristics for vehicle routing problem: A survey and recent advances, 2023.

[72] Ruifan Liu, Hyo-Sang Shin, Minguk Seo, and Antonios Tsourdos. *Delivery Route Planning for Unmanned Aerial System in Presence of Recharging Stations*.

[73] Ruifan Liu, Hyo-Sang Shin, and Antonios Tsourdos. Edge-enhanced attentions for drone delivery in presence of winds and recharging stations. *Journal of Aerospace Information Systems*, 20(4):216–228, 2023.

[74] Loggi. loggibud: Loggi benchmark for urban deliveries. https://github.com/loggi/loggibud, 2021.

[75] Rui Lu, Gao Huang, and Simon S. Du. On the power of multitask representation learning in linear mdp, 2021.

[76] Jia Luo, Chaofeng Li, Qinqin Fan, and Yuxin Liu. A graph convolutional encoder and multi-head attention decoder network for tsp via reinforcement learning. *Engineering Applications of Artificial Intelligence*, 112:104848, 2022.

[77] Qiang Ma, Suwen Ge, Danyang He, Darshan Thaker, and Iddo Drori. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning, 2019.

[78] Sephora Madjiheurem and Laura Toni. Representation learning on graphs: A reinforcement learning application, 2019.

[79] Clara Martins, Daniel Monteiro, and Gonçalo Pascoal. Solving large-scale instances of the capacitated vehicle routing problem, 2022. [Unpublished manuscript].

[80] Jason Mathers, Elena Craft, Marcelo Norsworthy, and Christina Wolfe. The green freight handbook, 2023. [Online; accessed 4-julho-2023].

[81] Daniel Merchán, Jatin Arora, Julian Pachon, Karthik Konduri, Matthias Winkenbach, Steven Parks, and Joseph Noszek. 2021 amazon last mile routing research challenge: Data set. *Transportation Science*, 0(0):null, 0.

[82] Astrid Merckling, Nicolas Perrin-Gilbert, Alex Coninx, and Stéphane Doncieux. Exploratory state representation learning. *Frontiers in Robotics and AI*, 9, 2022.

[83] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[84] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[85] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning, 2017.

[86] MohammadReza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takac. Reinforcement learning for solving the vehicle routing problem. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[87] OpenAI. Part 2: Kinds of rl algorithms — spinning up documentation. https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html, Nov 2018. [Online; accessed 5-February-2023].

[88] Joel Oren, Chana Ross, Maksym Lefarov, Felix Richter, Ayal Taitler, Zohar Feldman, Dotan Di Castro, and Christian Daniel. Solo: Search online, learn offline for combinatorial optimization problems. In *Vol. 12 No. 1 (2021): Fourteenth International Symposium on Combinatorial Search*, pages 97–105, Jinan University, Guangzhou, China, 2021. Association for the Advancement of Artificial Intelligence.

[89] Wenbin Ouyang, Yisen Wang, Shaochen Han, Zhejian Jin, and Paul Weng. Improving generalization of deep reinforcement learning-based tsp solvers. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 01–08, 2021.

[90] Sherjil Ozair, Corey Lynch, Yoshua Bengio, Aaron van den Oord, Sergey Levine, and Pierre Sermanet. Wasserstein dependency measure for representation learning, 2019.

[91] Julian Pachon, Daniel Merchán, Yossi Sheffi, and Matthias Winkenbach. About the Challenge | Amazon Last-Mile Routing Research Challenge. https://routingchallenge.mit.edu/about-the-challenge/, Sep 2022. [Online; accessed 5-February-2023].

[92] Junyoung Park, Sanjar Bakhtiyar, and Jinkyoo Park. Schedulenet: Learn to solve multi-agent scheduling problems with reinforcement learning, 2021.

[93] 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.

[94] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[95] Bo Peng, Jiahai Wang, and Zizhen Zhang. A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. In Kangshun Li, Wei Li, Hui Wang, and Yong Liu, editors, *Artificial Intelligence Algorithms and Applications*, pages 636–650, Singapore, 2020. Springer Singapore.

[96] J. Peters. Policy gradient methods. *Scholarpedia*, 5(11):3698, 2010. revision #137199.

[97] Julie Poullet. Leveraging machine learning to solve the vehicle routing problem with time windows. Master's thesis, Massachusetts Institute of Technology, Massachusetts, 2020. https://hdl.handle.net/1721.1/127285.

[98] Jacob Rafati. *Learning Representations in Reinforcement Learning*. PhD thesis, University of California, Merced, 04 2019.

[99] Ievgen Redko, Emilie Morvant, Amaury Habrard, Marc Sebban, and Younes Bennani. *Advances in domain adaptation theory*. Elsevier, 2019.

[100] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier Science Inc., USA, 2006.

[101] Franz Rothlauf. *Optimization Methods*, pages 45–102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[102] Alex Salgo, Jeremy Banks, and Francois Rivest. Exploring decision support systems in task scheduling. In *Proceedings of the 31st Annual International Conference on Computer Science and Software Engineering*, CASCON '21, page 184–189, USA, 2021. IBM Corp.

[103] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2015.

[104] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[105] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

[106] Jagdeep Singh, Sanjay Kumar Dhurandher, Isaac Woungang, and Telex Magloire N. Ngatched. Multi-agent reinforcement learning based approach for vehicle routing problem. In Telex Magloire Ngatched Nkouatchah, Isaac Woungang, Jules-Raymond Tapamo, and Serestina Viriri, editors, *Pan-African Artificial Intelligence and Smart Systems*, pages 411–422, Cham, 2023. Springer Nature Switzerland.

[107] Pawel Sitek and Jarosław Wikarek. Capacitated vehicle routing problem with pick-up and alternative delivery (cvrppad): model and implementation using hybrid approach. *Annals of Operations Research*, 273(1):257–277, Feb 2019.

[108] Ahmed Stohy, Heba-Tullah Abdelhakam, Sayed Ali, Mohammed Elhenawy, Abdallah A. Hassan, Mahmoud Masoud, Sebastien Glaser, and Andry Rakotonirainy. Hybrid pointer networks for traveling salesman problems optimization. *PLOS ONE*, 16(12):1–17, 12 2021.

[109] Niklas Strauss, David Winkel, Max Berrendorf, and Matthias Schubert. Reinforcement learning for multi-agent stochastic resource collection. In Massih-Reza Amini, Stéphane Canu, Asja Fischer, Tias Guns, Petra Kralj Novak, and Grigorios Tsoumakas, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 200–215, Cham, 2023. Springer Nature Switzerland.

[110] Nasrin Sultana, Jeffrey Chan, Tabinda Sarwar, and A. K. Qin. Sample-efficient, exploration-based policy optimisation for routing problems, 2022.

[111] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[112] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[113] Quinlan Sykora, Mengye Ren, and Raquel Urtasun. Multi-agent routing value iteration network. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9300–9310. PMLR, 13–18 Jul 2020.

[114] Matthew E Taylor and Peter Stone. Representation transfer for reinforcement learning. In *AAAI Fall Symposium: Computational Approaches to Representation Change during Learning and Development*, pages 78–85, 2007.

[115] Matthew E. Taylor and Peter Stone. Towards reinforcement learning representation transfer. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, New York, NY, USA, 2007. Association for Computing Machinery.

[116] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

[117] Christian Tilk, Katharina Olkis, and Stefan Irnich. The last-mile vehicle routing problem with delivery options. *OR Spectrum*, 43(4):877–904, Dec 2021.

[118] Paolo Toth and Daniele Vigo. Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, 123(1):487–512, 2002.

[119] Martijn van Otterlo and Marco Wiering. *Reinforcement Learning and Markov Decision Processes*, pages 3–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[120] Guido Van Rossum. *The Python Library Reference, release 3.8.2*. Python Software Foundation, 2020.

[121] José Manuel Vera and Andres G. Abad. Deep reinforcement learning for routing a heterogeneous fleet of vehicles. In *2019 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, pages 1–6, 2019.

[122] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[123] Han Wang. Emergent representations in reinforcement learning and their properties. Master's thesis, University of Alberta, 2020.

[124] Han Wang, Erfan Miahi, Martha White, Marlos C. Machado, Zaheer Abbas, Raksha Kumaraswamy, Vincent Liu, and Adam White. Investigating the properties of neural network representations in reinforcement learning, 2022.

[125] Qi Wang. Alpha-t: Learning to traverse over graphs with analphazero-inspired self-play framework, 2021. https://www.researchsquare.com/article/rs-415344/v1.

[126] Qi Wang, Yongsheng Hao, and Jie Cao. Learning to traverse over graphs with a monte carlo tree search-based self-play framework. *Engineering Applications of Artificial Intelligence*, 105:104422, 2021.

[127] Qi Wang, Yuqing He, and Chunlei Tang. Mastering construction heuristics with self-play deep reinforcement learning. *Neural Computing and Applications*, 35(6):4723–4738, Feb 2023.

[128] Qi Wang, Kenneth H. Lai, and Chunlei Tang. Solving combinatorial optimization problems over graphs with bert-based deep reinforcement learning. *Information Sciences*, 619:930–946, 2023.

[129] Qi Wang and Chunlei Tang. Deep reinforcement learning for transportation network combinatorial optimization: A survey. *Knowledge-Based Systems*, 233:107526, 2021.

[130] 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*, pages 1–15, 2022.

[131] Zhaodong Wang, Zhiwei Qin, Xiaocheng Tang, Jieping Ye, and Hongtu Zhu. Deep reinforcement learning with knowledge transfer for online rides order dispatching. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 617–626, 2018.

[132] Théophane Weber, Sébastien Racanière, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning, 2017.

[133] Jiayi Weng, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Yi Su, Hang Su, and Jun Zhu. Tianshou: A highly modularized deep reinforcement learning library. *Journal of Machine Learning Research*, 23(267):1–6, 2022.

[134] Eric Wiewiora. *Reward Shaping*, pages 863–865. Springer US, Boston, MA, 2010.

[135] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning, May 1992.

[136] Chen Wu, Yin Song, Verdi March, and Eden Duthie. Learning from drivers to tackle the amazon last mile routing research challenge, 2022.

[137] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6861–6871. PMLR, 09–15 Jun 2019.

[138] Guohua Wu, Mingfeng Fan, Jianmai Shi, and Yanghe Feng. Reinforcement learning based truck-and-drone coordinated delivery. *IEEE Transactions on Artificial Intelligence*, pages 1–1, 2021.

[139] Guojin Wu, Zizhen Zhang, Hong Liu, and Jiahai Wang. Solving time-dependent traveling salesman problem with time windows with deep reinforcement learning. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 558–563, 2021.

[140] Yaoxin Wu, Jianan Zhou, Yunwen Xia, Xianli Zhang, Zhiguang Cao, and Jie Zhang. Neural airport ground handling. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–15, 2023.

[141] Ali Yaddaden, Sebastien Harispe, and Michel Vasquez. Is transfer learning helpful for neural combinatorial optimization applied to vehicle routing problems? *COMPUTING AND INFORMATICS*, 41(1):172–190, Apr. 2022.

[142] Yimo Yan, Andy H.F. Chow, Chin Pang Ho, Yong-Hong Kuo, Qihao Wu, and Chengshuo Ying. Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities. *Transportation Research Part E: Logistics and Transportation Review*, 162:102712, 2022.

[143] Hua Yang, Minghao Zhao, Lei Yuan, Yang Yu, Zhenhua Li, and Ming Gu. Memory-efficient transformer-based network model for traveling salesman problem. *Neural Networks*, 161:589–597, 2023.

[144] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Reinforcement learning with prototypical representations, 2021.

[145] Jiaming Yin, Weixiong Rao, and Chenxi Zhang. Learning shortest paths on large dynamic graphs. In *2021 22nd IEEE International Conference on Mobile Data Management (MDM)*, pages 201–208, 2021.

[146] Hongyu Zang, Xin Li, Jie Yu, Chen Liu, Riashat Islam, Remi Tachet Des Combes, and Romain Laroche. Behavior prior representation learning for offline reinforcement learning, 2022.

[147] Amy Zhang, Rowan Thomas McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2021.

[148] Cong Zhang, Yaoxin Wu, Yining Ma, Wen Song, Zhang Le, Zhiguang Cao, and Jie Zhang. A review on learning to solve combinatorial optimisation problems in manufacturing. *IET Collaborative Intelligent Manufacturing*, 5(1):e12072, 2023.

[149] Ke Zhang, Fang He, Zhengchao Zhang, Xi Lin, and Meng Li. Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach. *Transportation Research Part C: Emerging Technologies*, 121:102861, 2020.

[150] Ke Zhang, Meng Li, Jiguang Wang, Yunxuan Li, and Xi Lin. A two-stage learning-based method for large-scale on-demand pickup and delivery services with soft time windows. *Transportation Research Part C: Emerging Technologies*, 151:104122, 2023.

[151] Ke Zhang, Xi Lin, and Meng Li. Graph attention reinforcement learning with flexible matching policies for multi-depot vehicle routing problems. *Physica A: Statistical Mechanics and its Applications*, 611:128451, 2023.

[152] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):11, nov 2019.

[153] Xuezhou Zhang, Yuda Song, Masatoshi Uehara, Mengdi Wang, Alekh Agarwal, and Wen Sun. Efficient reinforcement learning in block mdps: A model-free representation learning approach, 2022.

[154] Yongxin Zhang, Jiahai Wang, and Zizhen Zhang. Edge-based formulation with graph attention network for practical vehicle routing problem with time windows. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 01–08, 2022.

[155] Yongxin Zhang, Jiahai Wang, Zizhen Zhang, and Yalan Zhou. Modrl/d-el: Multiobjective deep reinforcement learning with evolutionary learning for multiobjective optimization. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[156] Zizhen Zhang, Hong Liu, MengChu Zhou, and Jiahai Wang. Solving dynamic traveling salesman problems with deep reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 34(4):2119–2132, 2023.

[157] Jiuxia Zhao, Minjia Mao, Xi Zhao, and Jianhua Zou. A hybrid of deep reinforcement learning and local search for the vehicle routing problems. *IEEE Transactions on Intelligent Transportation Systems*, 22(11):7208–7218, 2021.

[158] Tao Zhou, M.Y. Law Kris, Douglas Creighton, and Changzhi Wu. Gmix: Graph-based spatial-temporal multi-agent reinforcement learning for dynamic electric vehicle dispatching system. *Transportation Research Part C: Emerging Technologies*, 144:103886, 2022.

[159] Tianyu Zhu, Xinli Shi, Xiangping Xu, and Jinde Cao. An accelerated end-to-end method for solving routing problems. *Neural Networks*, 164:535–545, 2023.

[160] Zhuangdi Zhu, Kaixiang Lin, Anil K. Jain, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey, 2020.

[161] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2021.

[162] Zefang Zong, Tao Feng, Tong Xia, Depeng Jin, and Yong Li. Deep reinforcement learning for demand driven services in logistics and transportation systems: A survey, 2021.

# Appendix A

# Preprocessing Pipeline

In order to effectively utilize the data from the datasets for the learning process of the RL agent, several steps need to be taken. Moreover, the data from both datasets needs to be in an identical form. Figure A.1 provides a comprehensive pipeline view.

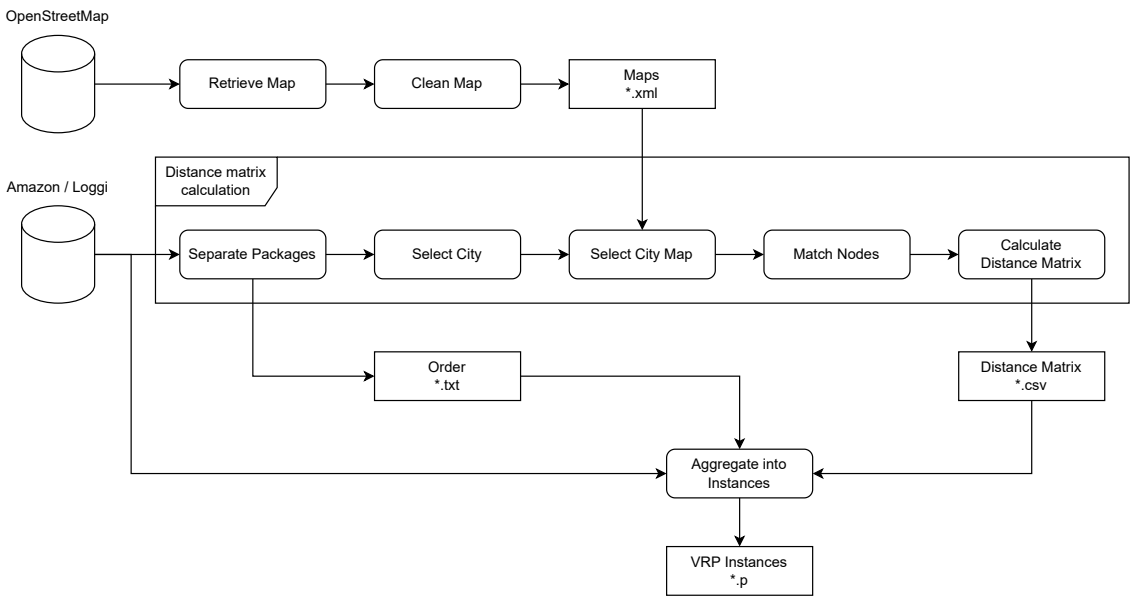


Figure A.1: Data Preprocessing Pipeline

The first step is obtaining the datasets. The Loggi dataset [74] can be obtained directly from the LoggiBUD Github repository, while the Amazon dataset [81] can be downloaded using the following command:

```
aws s3 ls -no-sign-request s3://amazon-last-mile-challenges/
```

The second step is obtaining and cleaning the city maps. Firstly, we retrieve a map for each city in the datasets from the OpenStreetMap [35]. Next, we clean each map, removing all the

nodes and edges outside the largest strongly connected component. Immediately afterwards, the map is stored in an XML file. Table A.1 describes each map.

- *Area (km$^2$)*: the city's area in square kilometres (km$^2$).
- *N Nodes*: the number of nodes in the city's map (after cleaning).
- *N Edges*: the number of edges in the city's map (after cleaning).

Table A.1: Description of each Map

| City | Area | N Nodes | N Edges |
|---|---|---|---|
| **Belém** | 8836.842 | 60475 | 71914 |
| **Brasília** | 10688.753 | 184661 | 215130 |
| **Rio de Janeiro** | 10909.896 | 375603 | 418913 |
| **Austin** | 5621.605 | 181060 | 195736 |
| **Boston** | 17474.827 | 1197090 | 1246074 |
| **Seattle** | 9612.950 | 271378 | 290920 |
| **Chicago** | 13355.364 | 682750 | 750833 |
| **Los Angeles** | 35670.848 | 1075238 | 1176532 |

The third step is calculating the distance matrices. We start by separating each package into its own delivery point (as referred to in Section 5.1). Afterwards, we identify the city and select the corresponding city map. Then, we match the depot and each delivery point with a node in the map. Lastly, we compute the distance between every pair of matched nodes and store it in a distance matrix (saved in a CSV file). This step uses code adapted from [79].

The final step is aggregating all the information and creating the instance files. We start by parsing the dataset files again, following the same file and package order as before, in order to match each instance with the correct distance matrix and each row and column with the proper delivery. Lastly, we combine the parsed data with the already created distance matrices to generate the instance files (stored as `pickle` [120] files).

Listing A.1 showcases the relevant instance fields described below using Python [120].

- `demands`: demand for each customer.
- `distance_matrix`: pair-wise distance between all the pairs of locations. The first location is the depot, followed by all the delivery points (customer locations). It is constructed as a matrix with the rows as the origins and the columns as the destinations. Before storing, it is flattened.
- `vehicles`: capacities for each vehicle. Since we consider a single vehicle, this list only contains one value.
- `start_time`: starting time of the first trip.

- `travel_times`: time needed for the vehicle to travel between every pair of locations. Similarly to the distance matrix, the first location is the depot, followed by all the delivery points. On top of that, it is also constructed as a matrix (with the rows corresponding to the origins and the columns corresponding to the destinations) and, later, flattened.
- `service_times`: time needed to load the vehicle at the depot, followed by the time needed to deliver the package at each delivery point.
- `time_windows`: requested time window for delivery to each customer. To avoid penalties regarding the infringement of this constraint, the vehicle would need to arrive at the specified location between the `start` and `end` times.

```
class Instance:
    demands         : List[float]
    vehicles        : List[float]
    distance_matrix : List[float]
    start_time      : float | None
    travel_times    : List[float] | None
    service_times   : List[float] | None
    time_windows    : List[TimeWindow] | None

    class TimeWindow:
    start: float | None
    end  : float | None
```

Listing A.1: VRP Instance

# Appendix B

# Hyperparameters

This chapter details the exact hyperparameters used in each of the experiments. For ease of understanding, we present a structure similar to the one adopted to present the results in Chapter 5.

## B.1 Core Configuration

In this project, we used the tianshou library [133] to facilitate the implementation. Therefore, some of the hyperparameters described here coincide or have a direct mapping with the ones described by this library. Our hyperparameters include:

- *Run ID*: ID used internally to determine which model to transfer the knowledge from when using Transfer Learning.
- *Algorithm*: the algorithm employed to train the RL agent. All our agents are trained with the REINFORCE algorithm.
- *Environment*: the environment in which the agent operates.
- *Fleet Type*: type of fleet managed. In accordance with the datasets, the fleet is homogeneous and infinite, meaning all the vehicles have equal capacity, and there are no restrictions on the number of trips those vehicles can make to serve customers.
- *Node Features*: node features used to represent the environment.
- *Mask*: mask scheme adopted to limit the agent's possible actions.
- *Reward Function*: the reward function employed to guide the learner.
- *Source ID for Transfer Learning*: ID of a stored agent whose model will serve as the source in the knowledge transfer. When left empty, no Transfer Learning is employed.

- *Resume Training*: whether or not to resume training from an existing checkpoint. To ensure reproducibility, our agents are trained in a single run without resuming training. Therefore, this value is always `False`.
- *Resume Epoch*: epoch used to resume training or transfer knowledge.
- *Dataset*: dataset exploited to train and test the agent's decision-making skills. Due to the huge graph size of the instances in the Loggi dataset, most of our experiments run on the Amazon dataset.
- *Train Instances*: IDs of the instances from the datasets used during the training phases (as obtained by our data preprocessing pipeline explored in Section A).
- *Test Instances*: IDs of the instances from the datasets used during the testing phases (as obtained by our data preprocessing pipeline explored in Section A).
- *Seed*: seed used to initialise all the random generators.
- *N Epochs*: number of training epochs.
- *Steps per Epoch*: number of steps or transitions collected per training epoch.
- *Max Steps per Episode*: maximum number of steps per episode (only applicable when the *Invalid* mask is not in use).
- *Steps per Collect*: number of steps or transitions collected before performing a network update.
- *Repeat per Collect*: number of times the policy needs to learn from each batch of data.
- *N Training*: number of parallel environments used for training.
- *N Testing*: number of episodes per policy evaluation.
- *Same Seed*: whether or not the same seed should be used to initialise all the environments.
- *Epochs per Instance*: number of epochs to train using the same training instance.
- *Buffer Size*: the size of the replay buffer. The replay buffer stores and manages the experiences of an agent during its interactions with an environment.
- *Batch Size*: number of experiences processed simultaneously (batch).
- *Learning Rate ($\alpha$)*: the learning rate.
- *Discount Factor ($\gamma$)*: the discount factor.
- *Device*: either *CPU* or *GPU* indicating whether we employ *CPU* or *GPU* for training.
- *Checkpoint Interval*: number of epochs between each checkpoint saved.

- *Graph Layer Type*: specific layer adopted as the Graph Layer.
- *Linear Layer Type*: specific layer adopted as the Linear Layer.
- *Graph Activation Type*: activation function applied in the Graph Layer.
- *Linear Layer Type*: activation function applied in the Linear Layer.
- *Graph Layers Hidden Sizes*: a list containing the number of neurons used in each hidden graph layer.
- *Linear Layers Hidden Sizes*: a list containing the number of neurons used in each hidden linear layer.
- *Optimiser*: optimiser that will update the parameters based on the computed gradients.

As previously described in Section 5.2, our Graph Neural Network configuration is identical to the one used by the authors of [39]. Therefore, we use a single Graph Convolutional Network layer (`GCNConv` [53]) from the `pytorch-geometric` package [33] and three Linear layers (`Linear`) from the pytorch package [94], with two hidden layers of 32 neurons each. Additionally, `ReLU` is used as the activation function in all layers except the last, which does not use an activation function, and `Adam` is used as the optimiser.

When the Amazon Dataset is used, the Train Instances correspond to [0, 1094] $\setminus$ {246, 316, 380, 651, 782} and the Test Instances correspond to [1095, 1096, 1097, 1098, 1099]. When the Loggi Dataset is used, the Train Instances correspond to [499, 506, 483, 476, 459, 466, 475, 363, 438, 484, 490, 436, 474, 507, 430, 457, 489, 451, 477, 443] and the Test Instances correspond to [481, 442, 386, 449, 497].

Additionally,

- *(*1)* represents the node features used in the Discount Factor Tuning experiment in the *CVRP* environment. It includes [*location*, *depot*, *demand*, *current_capacity*, *cost*, *can_serve*, mask].
- *(*2)* represents the node features used in the experiments in the *CVRP* environment. It includes [*location*, *depot*, *demand*, *current_capacity*, *next_capacity*, *fuel_consumption*, *cost*].
- *(*3)* represents the node features used in the experiments in the *CVRPSTW* environment. It includes [*location*, *depot*, *demand*, *current_capacity*, *next_capacity*, *fuel_consumption*, *cost*, *current_time*, *travel_time*, *service_time*, *start_tw*, *end_tw*, *tw_error*, *time_after_service*].

The model used in TL corresponds to the checkpoint that produced the highest reward (in that epoch).

In the graph, the number of epochs presented is the smaller number of epochs trained in each group of runs.

### B.1.1 Computer Specifications

Due to time restrictions, multiple computers were used in the experiments. To allow reproduction of the results, the hardware and software specifications used in this project's experiments is presented below.

Table B.1: Computer #1 Specifications

| | |
|---|---|
| **CPU** | Intel (R) Core (TM) i7-7700k CPU @ 4.2GHz |
| **RAM** | 32GB (+ 2GB swap) |
| **GPU** | NVIDA GeForce GTX 1080 |
| **GPU Memory** | 8GB |
| **Operating System** | Ubuntu 18.04.6 LTS |
| **Python Version** | 3.10.11 |
| **Pytorch Version** | 2.0.1+cu117 |
| **Pytorch Geometric Version** | 2.3.1 |
| **CUDA Version** | 12.1 |
| **Driver Version** | 530.41.03 |
| **Desktop or Laptop** | Desktop |

Table B.2: Computer #2 Specifications

| | |
|---|---|
| **CPU** | Intel (R) Core (TM) i7-7700k CPU @ 4.2GHz |
| **RAM** | 16GB (+ 8GB swap) |
| **GPU** | NVIDIA GeForce GTX 1080 |
| **GPU Memory** | 8GB |
| **Operating System** | Mint 19.2 Tina |
| **Python Version** | 3.10.11 |
| **Pytorch Version** | 2.0.1+cu117 |
| **Pytorch Geometric Version** | 2.3.1 |
| **CUDA Version** | 12.1 |
| **Driver Version** | 530.30.02 |
| **Desktop or Laptop** | Desktop |

### B.1.2 Discount Factor Tuning

## B.2 Capacitated Vehicle Routing Problem (CVRP)

### B.2.1 The Masking Mechanism

### B.2.2 Node Features

Table B.3: Computer #3 Specifications

| | |
|---|---|
| **CPU** | Intel (R) Core (TM) i7-4790k CPU @ 4GHz |
| **RAM** | 32GB (+ 4GB swap) |
| **GPU** | NVIDIA GeForce GTX 3060 Lite Hash Rate |
| **GPU Memory** | 12GB |
| **Operating System** | Ubuntu 22.04.2 LTS |
| **Python Version** | 3.10.6 |
| **Pytorch Version** | 2.0.1+cu117 |
| **Pytorch Geometric Version** | 2.3.1 |
| **CUDA Version** | 12.1 |
| **Driver Version** | 530.30.02 |
| **Desktop or Laptop** | Desktop |

Table B.4: Computer #4 Specifications

| | |
|---|---|
| **CPU** | 13th Gen Intel (R) Core (TM) i9-13900k |
| **RAM** | 64GB (+ 8GB swap) |
| **GPU** | NVIDIA GeForce GTX 4070 |
| **GPU Memory** | 12GB |
| **Operating System** | Ubuntu 23.04 |
| **Python Version** | 3.11.2 |
| **Pytorch Version** | 2.0.1+cu117 |
| **Pytorch Geometric Version** | 2.3.1 |
| **CUDA Version** | 12.0 |
| **Driver Version** | 525.125.06 |
| **Desktop or Laptop** | Desktop |

Table B.5: Computer #5 Specifications

| | |
|---|---|
| **CPU** | Intel (R) Core (TM) i7-8750H CPU @ 2.2GHz |
| **RAM** | 16GB (+ 16GB swap) |
| **GPU** | NVIDIA GeForce GTX 1050 Mobile |
| **GPU Memory** | 4GB |
| **Operating System** | Ubuntu 22.04.02 LTS |
| **Python Version** | 3.10.6 |
| **Pytorch Version** | 2.0.1+cu117 |
| **Pytorch Geometric Version** | 2.3.1 |
| **CUDA Version** | 12.1 |
| **Driver Version** | 530.30.02 |
| **Desktop or Laptop** | Laptop |

Table B.6: Parameters Used in the Discount Factor Tuning Experiment in the CVRP Environment

| **Parameter** | **Value** | | | | |
|---|---|---|---|---|---|
| Label in the Graph | 1 | 0.99 | 0.98 | 0.97 | 0.9 |
| Run ID | t1 | t5 | t8 | t3 | t2 |
| Algorithm | REINFORCE | REINFORCE | REINFORCE | REINFORCE | REINFORCE |
| Environment | CVRP | CVRP | CVRP | CVRP | CVRP |
| Node Features | *(*1)* | *(*1)* | *(*1)* | *(*1)* | *(*1)* |
| Mask | Invalid | Invalid | Invalid | Invalid | Invalid |
| Reward Function | Distance Norm. | Distance Norm. | Distance Norm. | Distance Norm. | Distance Norm. |
| Source ID for TL | - | - | - | - | - |
| Dataset | Amazon | Amazon | Amazon | Amazon | Amazon |
| Seed | 42 | 42 | 42 | 42 | 42 |
| N Epochs | 500 | 500 | 500 | 500 | 500 |
| Steps per Epoch | 1024 | 1024 | 1024 | 1024 | 1024 |
| Max Steps per Episode | None | None | None | None | None |
| Steps per Collect | 512 | 512 | 512 | 512 | 512 |
| Repeat per Collect | 1 | 1 | 1 | 1 | 1 |
| N Training | 4 | 4 | 4 | 4 | 4 |
| N Testing | 5 | 5 | 5 | 5 | 5 |
| Same Seed | True | True | True | True | True |
| Epochs per Instance | 1 | 1 | 1 | 1 | 1 |
| Buffer Size | 1024 | 1024 | 1024 | 1024 | 1024 |
| Batch Size | 64 | 64 | 64 | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 | 0.0001 | 0.0001 | 0.0001 |
| Discount Factor | 1 | 0.99 | 0.98 | 0.97 | 0.9 |
| Device | GPU | GPU | GPU | GPU | GPU |
| Checkpoint Interval | 1 | 1 | 1 | 1 | 1 |
| Computer | #1 | #4 | #4 | #2 | #4 |

Table B.7: Parameters Used in the Masking Mechanism Experiment in the CVRP Environment

| **Parameter** | **Value** | | |
|---|---|---|---|
| Label in the Graph | Invalid | Noop | None |
| Run ID | t4 | t6 | t7 |
| Algorithm | REINFORCE | REINFORCE | REINFORCE |
| Environment | CVRP | CVRP | CVRP |
| Node Features | *(*2)* | *(*2)* | *(*2)* |
| Mask | Invalid | Noop | None |
| Reward Function | Distance Norm. | Distance Norm. | Distance Norm. |
| Source ID for TL | - | - | - |
| Dataset | Amazon | Amazon | Amazon |
| Seed | 42 | 42 | 42 |
| N Epochs | 5000 | 5000 | 5000 |
| Steps per Epoch | 1024 | 1024 | 1024 |
| Max Steps per Episode | None | 512 | 512 |
| Steps per Collect | 512 | 512 | 512 |
| Repeat per Collect | 1 | 1 | 1 |
| N Training | 4 | 4 | 4 |
| N Testing | 5 | 5 | 5 |
| Same Seed | True | True | True |
| Epochs per Instance | 1 | 1 | 1 |
| Buffer Size | 1024 | 1024 | 1024 |
| Batch Size | 64 | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 | 0.98 |
| Device | GPU | GPU | GPU |
| Checkpoint Interval | 1 | 1 | 1 |
| Computer | #4 | #1 | #2 |

Table B.8: Parameters Used in the Node Features Experiment in the CVRP environment

| **Parameter** | **Value** |
|---|---|
| Algorithm | REINFORCE |
| Environment | CVRP |
| Mask | Invalid |
| Reward Function | Distance Norm. |
| Source ID for TL | - |
| Dataset | Amazon |
| Seed | 42 |
| N Epochs | 25 |
| Steps per Epoch | 1024 |
| Max Steps per Episode | None |
| Steps per Collect | 512 |
| Repeat per Collect | 1 |
| N Training | 4 |
| N Testing | 5 |
| Same Seed | True |
| Epochs per Instance | 1 |
| Buffer Size | 1024 |
| Batch Size | 64 |
| Learning Rate | 0.0001 |
| Discount Factor | 0.98 |
| Device | GPU |
| Checkpoint Interval | 1 |

Table B.9: Computers Used in the Node Features Experiment in the CVRP environment

| Node Features | | | | | | | Computer |
|---|---|---|---|---|---|---|---|
| *current_capacity* | *next_capacity* | *possible_capacity* | *can_serve* | *fuel_consumption* | *cost* | *mask* | Used |
| | | | | | | | #5 |
| X | | | | | | | #5 |
| | X | | | | | | #5 |
| | | X | | | | | #5 |
| | | | X | | | | #5 |
| | | | | X | | | #5 |
| | | | | | X | | #5 |
| | | | | | | X | #5 |
| X | X | | | | | | #5 |
| X | | X | | | | | #5 |
| X | | | X | | | | #5 |
| X | | | | X | | | #5 |
| X | | | | | X | | #5 |
| X | | | | | | X | #5 |
| | X | X | | | | | #5 |
| | X | | X | | | | #5 |
| | X | | | X | | | #5 |
| | X | | | | X | | #5 |
| | X | | | | | X | #5 |
| | | X | X | | | | #5 |
| | | X | | X | | | #5 |
| | | X | | | X | | #5 |
| | | X | | | | X | #5 |
| | | | X | X | | | #5 |
| | | | X | | X | | #5 |
| | | | X | | | X | #5 |
| | | | | X | X | | #5 |

Table B.9: Computers Used in the Node Features Experiment in the CVRP environment

| Node Features | | | | | | | Computer |
|---|---|---|---|---|---|---|---|
| *current_capacity* | *next_capacity* | *possible_capacity* | *can_serve* | *fuel_consumption* | *cost* | *mask* | Used |
| | | | | X | | X | #5 |
| | | | | | X | X | #5 |
| X | X | X | | | | | #5 |
| X | X | | X | | | | #1 |
| X | X | | | X | | | #1 |
| X | X | | | | X | | #1 |
| X | X | | | | | X | #1 |
| X | | X | X | | | | #1 |
| X | | X | | X | | | #1 |
| X | | X | | | X | | #1 |
| X | | X | | | | X | #1 |
| X | | | X | X | | | #1 |
| X | | | X | | X | | #1 |
| X | | | X | | | X | #1 |
| X | | | | X | X | | #1 |
| X | | | | X | | X | #1 |
| X | | | | | X | X | #1 |
| | X | X | X | | | | #1 |
| | X | X | | X | | | #1 |
| | X | X | | | X | | #1 |
| | X | X | | | | X | #1 |
| | X | | X | X | | | #1 |
| | X | | X | | X | | #1 |
| | X | | X | | | X | #5 |
| | X | | | X | X | | #5 |
| | X | | | X | | X | #5 |
| | X | | | | X | X | #4 |

Table B.9: Computers Used in the Node Features Experiment in the CVRP environment

| Node Features | | | | | | | Computer |
|---|---|---|---|---|---|---|---|
| *current_capacity* | *next_capacity* | *possible_capacity* | *can_serve* | *fuel_consumption* | *cost* | *mask* | Used |
| | | X | X | X | | | #4 |
| | | X | X | | X | | #4 |
| | | X | X | | | X | #4 |
| | | X | | X | X | | #4 |
| | | X | | X | | X | #4 |
| | | X | | | X | X | #4 |
| | | | X | X | X | | #4 |
| | | | X | X | | X | #4 |
| | | | X | | X | X | #4 |
| | | | | X | X | X | #4 |
| X | X | X | X | | | | #4 |
| X | X | X | | X | | | #4 |
| X | X | X | | | X | | #4 |
| X | X | X | | | | X | #4 |
| X | X | | X | X | | | #4 |
| X | X | | X | | X | | #4 |
| X | X | | X | | | X | #4 |
| X | X | | | X | X | | #4 |
| X | X | | | X | | X | #4 |
| X | X | | | | X | X | #4 |
| X | | X | X | X | | | #4 |
| X | | X | X | | X | | #4 |
| X | | X | X | | | X | #4 |
| X | | X | | X | X | | #4 |
| X | | X | | X | | X | #4 |
| X | | X | | | X | X | #4 |
| X | | | X | X | X | | #3 |

Table B.9: Computers Used in the Node Features Experiment in the CVRP environment

| Node Features | | | | | | | Computer |
|---|---|---|---|---|---|---|---|
| *current_capacity* | *next_capacity* | *possible_capacity* | *can_serve* | *fuel_consumption* | *cost* | *mask* | Used |
| X | | | X | X | | X | #3 |
| X | | | X | | X | X | #3 |
| X | | | | X | X | X | #3 |
| | X | X | X | X | | | #3 |
| | X | X | X | | X | | #3 |
| | X | X | X | | | X | #3 |
| | X | X | | X | X | | #3 |
| | X | X | | X | | X | #3 |
| | X | X | | | X | X | #3 |
| | X | | X | X | X | | #3 |
| | X | | X | X | | X | #3 |
| | X | | X | | X | X | #3 |
| | X | | | X | X | X | #3 |
| | | X | X | X | X | | #3 |
| | | X | X | X | | X | #3 |
| | | X | X | | X | X | #3 |
| | | X | | X | X | X | #3 |
| | | | X | X | X | X | #3 |
| X | X | X | X | X | | | #4 |
| X | X | X | X | | X | | #4 |
| X | X | X | X | | | X | #4 |
| X | X | X | | X | X | | #4 |
| X | X | X | | X | | X | #4 |
| X | X | X | | | X | X | #4 |
| X | X | | X | X | X | | #4 |
| X | X | | X | X | | X | #4 |
| X | X | | X | | X | X | #4 |

Table B.9: Computers Used in the Node Features Experiment in the CVRP environment

| Node Features | | | | | | | Computer |
|---|---|---|---|---|---|---|---|
| *current_capacity* | *next_capacity* | *possible_capacity* | *can_serve* | *fuel_consumption* | *cost* | *mask* | Used |
| X | X | | | X | X | X | #4 |
| X | | X | X | X | X | | #4 |
| X | | X | X | X | | X | #3 |
| X | | X | X | | X | X | #3 |
| X | | X | | X | X | X | #3 |
| X | | | X | X | X | X | #3 |
| | X | X | X | X | X | | #3 |
| | X | X | X | X | | X | #3 |
| | X | X | X | | X | X | #3 |
| | X | X | | X | X | X | #3 |
| | X | | X | X | X | X | #3 |
| | | X | X | X | X | X | #3 |
| X | X | X | X | X | X | | #3 |
| X | X | X | X | X | | X | #3 |
| X | X | X | X | | X | X | #3 |
| X | X | X | | X | X | X | #3 |
| X | X | | X | X | X | X | #3 |
| X | | X | X | X | X | X | #3 |
| | X | X | X | X | X | X | #3 |
| X | X | X | X | X | X | X | #3 |

### B.2.3 Reward Functions

Due to a sudden drop in performance from the *Distance* agent after epoch 995, the figures involving this agent represent the values until epoch 995 instead of 1000. Including this epoch would not allow for any meaningful comparison as all the lines would overlap as a constant (in the Distance and Fuel Consumption figures), with the exception of the *Distance* agent, which would overlap until epoch 995 and then skyrocket.

Table B.10: Parameters Used in the Reward Functions Experiment in the CVRP environment

| **Parameter** | **Value** | | | | |
|---|---|---|---|---|---|
| Label in the Graph | Distance Norm. | Distance | Fraction | Savings | Sparse (Dist. Norm.) |
| Run ID | t4 | r8 | r4 | r5 | r1 |
| Algorithm | REINFORCE | REINFORCE | REINFORCE | REINFORCE | REINFORCE |
| Environment | CVRP | CVRP | CVRP | CVRP | CVRP |
| Node Features | *(*2)* | *(*2)* | *(*2)* | *(*2)* | *(*2)* |
| Mask | Invalid | Invalid | Invalid | Invalid | Invalid |
| Reward Function | Distance Norm. | Distance | Fraction | Savings | Sparse (Dist. Norm.) |
| Source ID for TL | - | - | - | - | - |
| Dataset | Amazon | Amazon | Amazon | Amazon | Amazon |
| Seed | 42 | 42 | 42 | 42 | 42 |
| N Epochs | 5000 | 1000 | 1000 | 1000 | 1000 |
| Steps per Epoch | 1024 | 1024 | 1024 | 1024 | 1024 |
| Max Steps per Episode | None | None | None | None | None |
| Steps per Collect | 512 | 512 | 512 | 512 | 512 |
| Repeat per Collect | 1 | 1 | 1 | 1 | 1 |
| N Training | 4 | 4 | 4 | 4 | 4 |
| N Testing | 5 | 5 | 5 | 5 | 5 |
| Same Seed | True | True | True | True | True |
| Epochs per Instance | 1 | 1 | 1 | 1 | 1 |
| Buffer Size | 1024 | 1024 | 1024 | 1024 | 1024 |
| Batch Size | 64 | 64 | 64 | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 | 0.0001 | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 |
| Device | GPU | GPU | GPU | GPU | GPU |
| Checkpoint Interval | 1 | 1 | 1 | 1 | 1 |
| Computer | #4 | #4 | #3 | #3 | #3 |

## B.3 CVRP with Soft Time Windows (CVRPSTW)

### B.3.1 CVRP vs CVRPSTW

### B.3.2 Node Features

Table B.11: Parameters Used in the Reward Functions Experiment in the CVRP environment

| **Parameter** | **Value** | | |
|---|---|---|---|
| Label in the Graph | Distance Norm. | Fuel Cons. | Multi (D + F) |
| Run ID | t4 | r2 | r3 |
| Algorithm | REINFORCE | REINFORCE | REINFORCE |
| Environment | CVRP | CVRP | CVRP |
| Node Features | *(*2)* | *(*2)* | *(*2)* |
| Mask | Invalid | Invalid | Invalid |
| Reward Function | Distance Norm. | Fuel Cons. | Multi (D + E) |
| Source ID for TL | - | - | - |
| Dataset | Amazon | Amazon | Amazon |
| Seed | 42 | 42 | 42 |
| N Epochs | 5000 | 1000 | 1000 |
| Steps per Epoch | 1024 | 1024 | 1024 |
| Max Steps per Episode | None | None | None |
| Steps per Collect | 512 | 512 | 512 |
| Repeat per Collect | 1 | 1 | 1 |
| N Training | 4 | 4 | 4 |
| N Testing | 5 | 5 | 5 |
| Same Seed | True | True | True |
| Epochs per Instance | 1 | 1 | 1 |
| Buffer Size | 1024 | 1024 | 1024 |
| Batch Size | 64 | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 | 0.98 |
| Device | GPU | GPU | GPU |
| Checkpoint Interval | 1 | 1 | 1 |
| Computer | #4 | #3 | #3 |

Table B.12: Parameters Used in the Reward Functions Experiment in the CVRP environment

| **Parameter** | **Value** | |
|---|---|---|
| Label in the Graph | Dense | Sparse |
| Run ID | t7 | tx |
| Algorithm | REINFORCE | REINFORCE |
| Environment | CVRP | CVRP |
| Node Features | *(*2)* | *(*2)* |
| Mask | None | None |
| Reward Function | Distance Norm. | Sparse (Dist. Norm.) |
| Source ID for TL | - | - |
| Dataset | Amazon | Amazon |
| Seed | 42 | 42 |
| N Epochs | 5000 | 1000 |
| Steps per Epoch | 1024 | 1024 |
| Max Steps per Episode | 512 | 512 |
| Steps per Collect | 512 | 512 |
| Repeat per Collect | 1 | 1 |
| N Training | 4 | 4 |
| N Testing | 5 | 5 |
| Same Seed | True | True |
| Epochs per Instance | 1 | 1 |
| Buffer Size | 1024 | 1024 |
| Batch Size | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 |
| Device | GPU | GPU |
| Checkpoint Interval | 1 | 1 |
| Computer | #2 | #4 |

Table B.13: Parameters Used in the Experiment comparing the CVRP and the CVRPSTW environments

| **Parameter** | **Value** | |
|---|---|---|
| Label in the Graph | CVRP | CVRPSTW |
| Run ID | t4 | t0 |
| Algorithm | REINFORCE | REINFORCE |
| Environment | CVRP | CVRPSTW |
| Node Features | *(*2)* | *(*3)* |
| Mask | Invalid | Invalid |
| Reward Function | Distance Norm. | TW Error (D) |
| Source ID for TL | - | - |
| Dataset | Amazon | Amazon |
| Seed | 42 | 42 |
| N Epochs | 5000 | 5000 |
| Steps per Epoch | 1024 | 1024 |
| Max Steps per Episode | None | None |
| Steps per Collect | 512 | 512 |
| Repeat per Collect | 1 | 1 |
| N Training | 4 | 4 |
| N Testing | 5 | 5 |
| Same Seed | True | True |
| Epochs per Instance | 1 | 1 |
| Buffer Size | 1024 | 1024 |
| Batch Size | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 |
| Device | GPU | GPU |
| Checkpoint Interval | 1 | 1 |
| Computer | #4 | #4 |

Table B.14: Parameters Used in the Node Features Experiment in the CVRPSTW environment

| **Parameter** | **Value** |
|---|---|
| Algorithm | REINFORCE |
| Environment | CVRPSTW |
| Mask | Invalid |
| Reward Function | TW Error (D) |
| Source ID for TL | - |
| Dataset | Amazon |
| Seed | 42 |
| N Epochs | 25 |
| Steps per Epoch | 1024 |
| Max Steps per Episode | None |
| Steps per Collect | 512 |
| Repeat per Collect | 1 |
| N Training | 4 |
| N Testing | 5 |
| Same Seed | True |
| Epochs per Instance | 1 |
| Buffer Size | 1024 |
| Batch Size | 64 |
| Learning Rate | 0.0001 |
| Discount Factor | 0.98 |
| Device | GPU |
| Checkpoint Interval | 1 |

Table B.15: Computers Used in the Node Features Experiment in the CVRPSTW environment

| Node Features | | | | | | Computer |
|---|---|---|---|---|---|---|
| *arrival_time* | *time_after_service* | *inside_tw* | *tw_error* | *after_tw_start* | *before_tw_end* | Used |
| | | | | | | #3 |
| X | | | | | | #3 |
| | X | | | | | #4 |
| | | X | | | | #3 |
| | | | X | | | #3 |
| | | | | X | | #3 |
| | | | | | X | #3 |
| X | X | | | | | #3 |
| X | | X | | | | #3 |
| X | | | X | | | #3 |
| X | | | | X | | #3 |
| X | | | | | X | #3 |
| | X | X | | | | #4 |
| | X | | X | | | #4 |
| | X | | | X | | #4 |
| | X | | | | X | #4 |
| | | X | X | | | #4 |
| | | X | | X | | #4 |
| | | X | | | X | #4 |
| | | | X | X | | #4 |
| | | | X | | X | #4 |
| | | | | X | X | #4 |
| X | X | X | | | | #4 |
| X | X | | X | | | #4 |
| X | X | | | X | | #4 |
| X | X | | | | X | #4 |
| X | | X | X | | | #4 |

Table B.15: Computers Used in the Node Features Experiment in the CVRPSTW environment

| Node Features | | | | | | Computer |
|---|---|---|---|---|---|---|
| *arrival_time* | *time_after_service* | *inside_tw* | *tw_error* | *after_tw_start* | *before_tw_end* | Used |
| X | | X | | X | | #4 |
| X | | X | | | X | #4 |
| X | | | X | X | | #4 |
| X | | | X | | X | #4 |
| X | | | | X | X | #4 |
| | X | X | X | | | #4 |
| | X | X | | X | | #4 |
| | X | X | | | X | #4 |
| | X | | X | X | | #4 |
| | X | | X | | X | #4 |
| | X | | | X | X | #4 |
| | | X | X | X | | #4 |
| | | X | X | | X | #4 |
| | | X | | X | X | #4 |
| | | | X | X | X | #4 |
| X | X | X | X | | | #4 |
| X | X | X | | X | | #4 |
| X | X | X | | | X | #4 |
| X | X | | X | X | | #4 |
| X | X | | X | | X | #4 |
| X | X | | | X | X | #4 |
| X | | X | X | X | | #4 |
| X | | X | X | | X | #4 |
| X | | X | | X | X | #4 |
| X | | | X | X | X | #4 |
| | X | X | X | X | | #4 |
| | X | X | X | | X | #4 |

Table B.15: Computers Used in the Node Features Experiment in the CVRPSTW environment

| Node Features | | | | | | Computer |
|---|---|---|---|---|---|---|
| *arrival_time* | *time_after_service* | *inside_tw* | *tw_error* | *after_tw_start* | *before_tw_end* | Used |
| | X | X | | X | X | #4 |
| | X | | X | X | X | #4 |
| | | X | X | X | X | #4 |
| X | X | X | X | X | | #4 |
| X | X | X | X | | X | #4 |
| X | X | X | | X | X | #4 |
| X | X | | X | X | X | #4 |
| X | | X | X | X | X | #4 |
| | X | X | X | X | X | #4 |
| X | X | X | X | X | X | #4 |

### B.3.3 Reward Functions

Table B.16: Parameters Used in the Reward Functions Experiment in the CVRPSTW environment

| **Parameter** | **Value** | |
|---|---|---|
| Label in the Graph | TW Error (D) | TW Failed (D) |
| Run ID | t0 | r0 |
| Algorithm | REINFORCE | REINFORCE |
| Environment | CVRPSTW | CVRPSTW |
| Node Features | *(*3)* | *(*3)* |
| Mask | Invalid | Invalid |
| Reward Function | TW Error (D) | TW Failed (D) |
| Source ID for TL | - | - |
| Dataset | Amazon | Amazon |
| Seed | 42 | 42 |
| N Epochs | 5000 | 5000 |
| Steps per Epoch | 1024 | 1024 |
| Max Steps per Episode | None | None |
| Steps per Collect | 512 | 512 |
| Repeat per Collect | 1 | 1 |
| N Training | 4 | 4 |
| N Testing | 5 | 5 |
| Same Seed | True | True |
| Epochs per Instance | 1 | 1 |
| Buffer Size | 1024 | 1024 |
| Batch Size | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 |
| Device | GPU | GPU |
| Checkpoint Interval | 1 | 1 |
| Computer | #4 | #4 |

## B.4 Exploring Transfer Learning

### B.4.1 Changing Dataset

### B.4.2 Removing the Mask

### B.4.3 Changing Environment

### B.4.4 Changing Reward Functions

Table B.17: Parameters Used in the Changing Dataset Experiment

| **Parameter** | **Value** | |
|---|---|---|
| Label in the Graph | No | Yes |
| Run ID | ty | d4 |
| Algorithm | REINFORCE | REINFORCE |
| Environment | CVRP | CVRP |
| Node Features | *(*2)* | *(*2)* |
| Mask | Invalid | Invalid |
| Reward Function | Distance Norm. | Distance Norm. |
| Source ID for TL | - | t4 |
| Dataset | Loggi | Loggi |
| Seed | 42 | 42 |
| N Epochs | 100 | 100 |
| Steps per Epoch | 4096 | 4096 |
| Max Steps per Episode | None | None |
| Steps per Collect | 512 | 512 |
| Repeat per Collect | 1 | 1 |
| N Training | 4 | 4 |
| N Testing | 5 | 5 |
| Same Seed | True | True |
| Epochs per Instance | 1 | 1 |
| Buffer Size | 1024 | 1024 |
| Batch Size | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 |
| Device | GPU | GPU |
| Checkpoint Interval | 1 | 1 |
| Computer | #4 | #4 |

Table B.18: Parameters Used in the Removing the Mask Experiment

| **Parameter** | **Value** | |
|---|---|---|
| Label in the Graph | No | Yes |
| Run ID | t7 | dl |
| Algorithm | REINFORCE | REINFORCE |
| Environment | CVRP | CVRP |
| Node Features | *(*2)* | *(*2)* |
| Mask | None | None |
| Reward Function | Distance Norm. | Distance Norm. |
| Source ID for TL | - | t4 |
| Dataset | Amazon | Amazon |
| Seed | 42 | 42 |
| N Epochs | 5000 | 100 |
| Steps per Epoch | 1024 | 1024 |
| Max Steps per Episode | 512 | 512 |
| Steps per Collect | 512 | 512 |
| Repeat per Collect | 1 | 1 |
| N Training | 4 | 4 |
| N Testing | 5 | 5 |
| Same Seed | True | True |
| Epochs per Instance | 1 | 1 |
| Buffer Size | 1024 | 1024 |
| Batch Size | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 |
| Device | GPU | GPU |
| Checkpoint Interval | 1 | 1 |
| Computer | #2 | #3 |

Table B.19: Parameters Used in the Changing Environment Experiment (Target is Easier than Source)

| **Parameter** | **Value** | |
|---|---|---|
| Label in the Graph | No | Yes |
| Run ID | t4 | dk |
| Algorithm | REINFORCE | REINFORCE |
| Environment | CVRP | CVRP |
| Node Features | *(*2)* | *(*2)* |
| Mask | Invalid | Invalid |
| Reward Function | Distance Norm. | Distance Norm. |
| Source ID for TL | - | r0 |
| Dataset | Amazon | Amazon |
| Seed | 42 | 42 |
| N Epochs | 5000 | 100 |
| Steps per Epoch | 1024 | 1024 |
| Max Steps per Episode | None | None |
| Steps per Collect | 512 | 512 |
| Repeat per Collect | 1 | 1 |
| N Training | 4 | 4 |
| N Testing | 5 | 5 |
| Same Seed | True | True |
| Epochs per Instance | 1 | 1 |
| Buffer Size | 1024 | 1024 |
| Batch Size | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 |
| Device | GPU | GPU |
| Checkpoint Interval | 1 | 1 |
| Computer | #4 | #4 |

Table B.20: Parameters Used in the Changing Environment Experiment (Target is Harder than Source)

| **Parameter** | **Value** | |
|---|---|---|
| Label in the Graph | No | Yes |
| Run ID | t0 | d3 |
| Algorithm | REINFORCE | REINFORCE |
| Environment | CVRPSTW | CVRPSTW |
| Node Features | *(*3)* | *(*3)* |
| Mask | Invalid | Invalid |
| Reward Function | TW Error (D) | TW Error (D) |
| Source ID for TL | - | t4 |
| Dataset | Amazon | Amazon |
| Seed | 42 | 42 |
| N Epochs | 5000 | 100 |
| Steps per Epoch | 1024 | 1024 |
| Max Steps per Episode | None | None |
| Steps per Collect | 512 | 512 |
| Repeat per Collect | 1 | 1 |
| N Training | 4 | 4 |
| N Testing | 5 | 5 |
| Same Seed | True | True |
| Epochs per Instance | 1 | 1 |
| Buffer Size | 1024 | 1024 |
| Batch Size | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 |
| Device | GPU | GPU |
| Checkpoint Interval | 1 | 1 |
| Computer | #4 | #4 |

Table B.21: Parameters Used in the Changing Reward Functions Experiment (Target = Fraction)

| **Parameter** | **Value** | |
|---|---|---|
| Label in the Graph | - | Distance Norm. |
| Run ID | r4 | d2 |
| Algorithm | REINFORCE | REINFORCE |
| Environment | CVRP | CVRP |
| Node Features | *(*2)* | *(*2)* |
| Mask | Invalid | Invalid |
| Reward Function | Fraction | Fraction |
| Source ID for TL | - | t4 |
| Dataset | Amazon | Amazon |
| Seed | 42 | 42 |
| N Epochs | 1000 | 100 |
| Steps per Epoch | 1024 | 1024 |
| Max Steps per Episode | None | None |
| Steps per Collect | 512 | 512 |
| Repeat per Collect | 1 | 1 |
| N Training | 4 | 4 |
| N Testing | 5 | 5 |
| Same Seed | True | True |
| Epochs per Instance | 1 | 1 |
| Buffer Size | 1024 | 1024 |
| Batch Size | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 |
| Device | GPU | GPU |
| Checkpoint Interval | 1 | 1 |
| Computer | #3 | #4 |

Table B.22: Parameters Used in the Changing Reward Functions Experiment (Target = Savings)

| **Parameter** | **Value** | |
|---|---|---|
| Label in the Graph | - | Distance Norm. |
| Run ID | r5 | dh |
| Algorithm | REINFORCE | REINFORCE |
| Environment | CVRP | CVRP |
| Node Features | *(*2)* | *(*2)* |
| Mask | Invalid | Invalid |
| Reward Function | Savings | Savings |
| Source ID for TL | - | t4 |
| Dataset | Amazon | Amazon |
| Seed | 42 | 42 |
| N Epochs | 1000 | 100 |
| Steps per Epoch | 1024 | 1024 |
| Max Steps per Episode | None | None |
| Steps per Collect | 512 | 512 |
| Repeat per Collect | 1 | 1 |
| N Training | 4 | 4 |
| N Testing | 5 | 5 |
| Same Seed | True | True |
| Epochs per Instance | 1 | 1 |
| Buffer Size | 1024 | 1024 |
| Batch Size | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 |
| Device | GPU | GPU |
| Checkpoint Interval | 1 | 1 |
| Computer | #3 | #3 |

Table B.23: Parameters Used in the Changing Reward Functions Experiment (Target = Distance)

| **Parameter** | **Value** | |
|---|---|---|
| Label in the Graph | - | Distance Norm. |
| Run ID | r8 | dm |
| Algorithm | REINFORCE | REINFORCE |
| Environment | CVRP | CVRP |
| Node Features | *(*2)* | *(*2)* |
| Mask | Invalid | Invalid |
| Reward Function | Distance | Distance |
| Source ID for TL | - | t4 |
| Dataset | Amazon | Amazon |
| Seed | 42 | 42 |
| N Epochs | 1000 | 100 |
| Steps per Epoch | 1024 | 1024 |
| Max Steps per Episode | None | None |
| Steps per Collect | 512 | 512 |
| Repeat per Collect | 1 | 1 |
| N Training | 4 | 4 |
| N Testing | 5 | 5 |
| Same Seed | True | True |
| Epochs per Instance | 1 | 1 |
| Buffer Size | 1024 | 1024 |
| Batch Size | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 |
| Device | GPU | GPU |
| Checkpoint Interval | 1 | 1 |
| Computer | #4 | #3 |

Table B.24: Parameters Used in the Changing Reward Functions Experiment (Target = Distance Norm.)

| **Parameter** | **Value** | | |
|---|---|---|---|
| Label in the Graph | - | Fuel Cons. | Multi (D + F) |
| Run ID | t4 | db | di |
| Algorithm | REINFORCE | REINFORCE | REINFORCE |
| Environment | CVRP | CVRP | CVRP |
| Node Features | *(*2)* | *(*2)* | *(*2)* |
| Mask | Invalid | Invalid | Invalid |
| Reward Function | Distance Norm. | Distance Norm. | Distance Norm. |
| Source ID for TL | - | r2 | r3 |
| Dataset | Amazon | Amazon | Amazon |
| Seed | 42 | 42 | 42 |
| N Epochs | 5000 | 100 | 100 |
| Steps per Epoch | 1024 | 1024 | 1024 |
| Max Steps per Episode | None | None | None |
| Steps per Collect | 512 | 512 | 512 |
| Repeat per Collect | 1 | 1 | 1 |
| N Training | 4 | 4 | 4 |
| N Testing | 5 | 5 | 5 |
| Same Seed | True | True | True |
| Epochs per Instance | 1 | 1 | 1 |
| Buffer Size | 1024 | 1024 | 1024 |
| Batch Size | 64 | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 | 0.98 |
| Device | GPU | GPU | GPU |
| Checkpoint Interval | 1 | 1 | 1 |
| Computer | #4 | #4 | #4 |

Table B.25: Parameters Used in the Changing Reward Functions Experiment (Target = Fuel Cons.)

| **Parameter** | **Value** | | |
|---|---|---|---|
| Label in the Graph | - | Distance Norm. | Multi (D + F) |
| Run ID | r2 | d0 | dj |
| Algorithm | REINFORCE | REINFORCE | REINFORCE |
| Environment | CVRP | CVRP | CVRP |
| Node Features | *(*2)* | *(*2)* | *(*2)* |
| Mask | Invalid | Invalid | Invalid |
| Reward Function | Fuel Cons. | Fuel Cons. | Fuel Cons. |
| Source ID for TL | - | t4 | r3 |
| Dataset | Amazon | Amazon | Amazon |
| Seed | 42 | 42 | 42 |
| N Epochs | 1000 | 100 | 100 |
| Steps per Epoch | 1024 | 1024 | 1024 |
| Max Steps per Episode | None | None | None |
| Steps per Collect | 512 | 512 | 512 |
| Repeat per Collect | 1 | 1 | 1 |
| N Training | 4 | 4 | 4 |
| N Testing | 5 | 5 | 5 |
| Same Seed | True | True | True |
| Epochs per Instance | 1 | 1 | 1 |
| Buffer Size | 1024 | 1024 | 1024 |
| Batch Size | 64 | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 | 0.98 |
| Device | GPU | GPU | GPU |
| Checkpoint Interval | 1 | 1 | 1 |
| Computer | #3 | #4 | #4 |

Table B.26: Parameters Used in the Changing Reward Functions Experiment (Target = Multi (D + F))

| **Parameter** | **Value** | | |
|---|---|---|---|
| Label in the Graph | - | Distance Norm. | Fuel Cons. |
| Run ID | r3 | d1 | dc |
| Algorithm | REINFORCE | REINFORCE | REINFORCE |
| Environment | CVRP | CVRP | CVRP |
| Node Features | *(*2)* | *(*2)* | *(*2)* |
| Mask | Invalid | Invalid | Invalid |
| Reward Function | Multi (D + E) | Multi (D + E) | Multi (D + E) |
| Source ID for TL | - | t4 | r2 |
| Dataset | Amazon | Amazon | Amazon |
| Seed | 42 | 42 | 42 |
| N Epochs | 1000 | 100 | 100 |
| Steps per Epoch | 1024 | 1024 | 1024 |
| Max Steps per Episode | None | None | None |
| Steps per Collect | 512 | 512 | 512 |
| Repeat per Collect | 1 | 1 | 1 |
| N Training | 4 | 4 | 4 |
| N Testing | 5 | 5 | 5 |
| Same Seed | True | True | True |
| Epochs per Instance | 1 | 1 | 1 |
| Buffer Size | 1024 | 1024 | 1024 |
| Batch Size | 64 | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 | 0.98 |
| Device | GPU | GPU | GPU |
| Checkpoint Interval | 1 | 1 | 1 |
| Computer | #3 | #4 | #4 |

Table B.27: Parameters Used in the Changing Reward Functions Experiment (Target = TW Error (D))

| **Parameter** | **Value** | |
|---|---|---|
| Label in the Graph | No | Yes |
| Run ID | t0 | de |
| Algorithm | REINFORCE | REINFORCE |
| Environment | CVRPSTW | CVRPSTW |
| Node Features | *(*3)* | *(*3)* |
| Mask | Invalid | Invalid |
| Reward Function | TW Error (D) | TW Error (D) |
| Source ID for TL | - | r0 |
| Dataset | Amazon | Amazon |
| Seed | 42 | 42 |
| N Epochs | 5000 | 100 |
| Steps per Epoch | 1024 | 1024 |
| Max Steps per Episode | None | None |
| Steps per Collect | 512 | 512 |
| Repeat per Collect | 1 | 1 |
| N Training | 4 | 4 |
| N Testing | 5 | 5 |
| Same Seed | True | True |
| Epochs per Instance | 1 | 1 |
| Buffer Size | 1024 | 1024 |
| Batch Size | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 |
| Device | GPU | GPU |
| Checkpoint Interval | 1 | 1 |
| Computer | #4 | #4 |

Table B.28: Parameters Used in the Changing Reward Functions Experiment (Target = TW Failed (D))

| **Parameter** | **Value** | |
|---|---|---|
| Label in the Graph | No | Yes |
| Run ID | r0 | dd |
| Algorithm | REINFORCE | REINFORCE |
| Environment | CVRPSTW | CVRPSTW |
| Node Features | *(*3)* | *(*3)* |
| Mask | Invalid | Invalid |
| Reward Function | TW Failed (D) | TW Failed (D) |
| Source ID for TL | - | t0 |
| Dataset | Amazon | Amazon |
| Seed | 42 | 42 |
| N Epochs | 5000 | 100 |
| Steps per Epoch | 1024 | 1024 |
| Max Steps per Episode | None | None |
| Steps per Collect | 512 | 512 |
| Repeat per Collect | 1 | 1 |
| N Training | 4 | 4 |
| N Testing | 5 | 5 |
| Same Seed | True | True |
| Epochs per Instance | 1 | 1 |
| Buffer Size | 1024 | 1024 |
| Batch Size | 64 | 64 |
| Learning Rate | 0.0001 | 0.0001 |
| Discount Factor | 0.98 | 0.98 |
| Device | GPU | GPU |
| Checkpoint Interval | 1 | 1 |
| Computer | #4 | #4 |