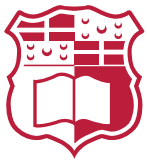
Multi-Vehicle Ride-Pooling System using Reinforcement Learning

Shaun Borg

Supervisor: Dr Josef Bajada

March 2024

Submitted in partial fulfilment of the requirements for the degree of M.Sc. in Artificial Intelligence.



L-Università ta' Malta
Faculty of Information & Communication Technology

Abstract

Ride-pooling services have surged in popularity in recent years due to them being more efficient, convenient, and cost-effective than alternative traditional methods such as taxis. Leveraging mobile technologies, ride pooling services use the location of drivers and customers to assign a shared vehicle to passengers travelling in the same direction, reducing the number of vehicles on the road and, therefore, helping reduce traffic congestion. Such services require algorithms that dynamically match passengers with nearby drivers and optimise routes. Ride-pooling can be considered a variant of the Vehicle Routing Problem (VRP), which is a combinatorial NP-hard problem as it involves finding an efficient set of routes for a fleet of vehicles to serve customers while satisfying constraints such as customers' time windows and vehicle capacity. Literature shows how the use of various metaheuristic algorithms to solve the VRP, such as tabu search (TS), can provide good solutions in terms of quality but suffer from scalability. With the recent advances in artificial intelligence, Reinforcement Learning (RL) is also being applied to capture the dynamic and stochastic nature of the VRP.

In this research, we propose to use RL to solve the Multi-Vehicle Routing for Ride-Pooling Problem (MVRRPP) and generate solutions faster than the traditional metaheuristic methods. This algorithm aims to optimise passenger allocation to vehicles and vehicle routing while minimising the overall waiting time, travel time and total driving distance. We first implemented a baseline algorithm that uses TS with an initial solution consisting of equally distributed customers along the routes to solve the MVRRPP and establish its performance. We then model the MVRRPP as an RL problem and solve it using the REINFORCE algorithm with a dynamic attention model consisting of a dynamic encoder-decoder architecture. The performance of this model was compared with the results achieved using TS. Finally, we evaluate the effect of using an RL solution as input for the TS algorithm.

The results of the three models showed that TS found higher-quality solutions than RL and TS with RL; however, its computational complexity resulted in longer computation times when solving large problem instances. Using RL involved a trade-off between solution quality and computation time, where it was quicker to find a solution even for problems on a large scale. On the other hand, performance using TS with RL showed minimal improvement except for reducing the distance travelled by vehicles, which suggests that the RL solution, used as the initial solution for TS, was in a region of the search space that had an inferior local optimum then the initial solution used in the TS without RL. The choice between TS and RL for solving the MVRRPP depends on the application's requirements and the problem's complexity.

Acknowledgements

I would like to thank my supervisor, Dr Josef Bajada for his guidance throughout this research. A special thanks goes to my family for their support.

Contents

List of Figures

List of Tables

List of Abbreviations

CSs Charging Stations.

DQN Deep Q-Network.

DRL Deep Reinforcement Learning.

EVRP Electric Vehicle Routing Problem.

EVs Electric Vehicles.

GA Genetic Algorithms.

GAT Graph Attention Network.

GNN Graph Neural Networks.

MC Monte Carlo.

MDP Markov Decision Processes.

MVRRPP Multi-Vehicle Routing for Ride-Pooling Problem.

RL Reinforcement Learning.

SA Simulated Annealing.

TD Temporal Difference.

TS Tabu Search.

VRP Vehicle Routing Problem.

1 Introduction

In recent years, ride-pooling has revolutionized how people commute, offering an efficient, convenient, cost-effective alternative to traditional methods such as taxis. Ride-pooling's popularity surged primarily due to the advancements in mobile technology and the widespread adoption of smartphones. Since most of the population owns a smartphone, ride-pooling services leverage mobile apps as the primary interface for users to request rides and track the location of their designated driver, making it convenient to access ride-pooling services anywhere and at any time. The integration of GPS (Global Positioning System) technology in smartphones has enabled ride-pooling platforms to match drivers with passengers based on their real-time locations. This convenience eliminates the need to hail taxis on the street or make phone calls to dispatch services, streamlining the process of requesting transportation. Sharing rides with other passengers through ride-pooling services can allow individuals to split transportation costs, making it more affordable than traditional taxis or private cars. Ride-pooling services assign a single vehicle for multiple passengers travelling in the same direction, reducing the number of vehicles on the road since the passengers are sharing a ride instead of each using their vehicle. By maximising vehicle occupancy, ride-pooling reduces the number of cars on the road during peak travel times, alleviating traffic congestion. With fewer vehicles, ride-pooling can help reduce overall emissions. Ride-pooling may also help alleviate city parking problems, which is especially beneficial in urban areas where parking spaces are limited and expensive.

1.1 Problem Definition

The Vehicle Routing Problem (VRP) is an optimisation problem used to determine the most efficient set of routes for a fleet of vehicles that serves a given set of customers while satisfying various constraints such as customer's time windows and vehicle capacity.

This research will tackle the Multi-Vehicle Routing for Ride-Pooling Problem (MVRRPP) which extends on the VRP to introduce additional complexities related to ride-pooling. The MVRRPP involves finding the optimal set of routes for a fleet of vehicles to serve multiple passengers with similar destinations using a single vehicle. Each passenger has a pickup location representing where the passenger requests a ride and a dropoff location representing the destination of that passenger. At the start of the problem, a fleet of vehicles is strategically placed in popular areas during peak hours or special events by the service provider to meet anticipated demand and reduce passenger waiting time proactively. The starting location of each vehicle is known as

the origin location, and it must return to its origin location after completing its route. Given a set of passenger requests, the objective of the problem is to maximise the utilisation of vehicles and the efficiency of their routes. The problem maximises the utilisation of vehicles by allocating a nearby vehicle to passengers with similar pickup and dropoff locations. The problem then maximises the efficiency of routes by optimising the sequence of pickups and dropoff locations by considering factors such as passenger waiting time and travel time and the distance travelled by the vehicle to minimise costs and environmental impact.

1.2 Motivation

With the growing popularity and demand for ride-pooling services, there is a need to develop algorithms that optimise routes in real-time, as this requires rapid processing of incoming requests and dynamically matching passengers with nearby drivers. Routes must be adapted in real-time to accommodate the addition of customers while minimising detours and waiting times for passengers. By efficiently assigning passengers to vehicles and determining optimal routes in real-time, such algorithms aim to reduce overall travel time, distance travelled, and traffic congestion by combining multiple passengers into one vehicle and, therefore, reducing carbon emissions.

While metaheuristic methods such as Tabu Search [1] and Genetic algorithms [2] have traditionally been used to solve the VRP, these methods often struggle to adapt to complex scenarios with many customers [3]. Metaheuristic methods often find difficulty in solving such problems within an efficient amount of time as the problem increases in demand [4]. However, recent reinforcement learning advancements present a promising approach for tackling VRP more adaptively and efficiently by learning techniques capable of capturing the dynamic and stochastic nature of the VRP [5].

1.3 Challenges

Due to the VRP being an NP-hard combinatorial optimisation problem [6], the number of possible solutions increases exponentially as the number of vehicles and passengers increases, encountering issues such as solution quality and slower running times when using metaheuristic methods [7, 8]. Recent advancements in Reinforcement Learning (RL) have shown impressive performance in solving combinational optimisation problems due to their ability to learn optimal decision-making policies through interactions with the problem environment [5].

However, solving the MVRRPP using RL poses several challenges. Such

challenges include defining the appropriate state representation for the MVRRPP in a way that allows the RL agent to capture the essential features such as the location of drivers and customers, vehicle capacity, distance travelled and waiting time, allowing it to learn how to optimise vehicle allocation and routing.

Implementing an RL algorithm with a dynamic attention model is also challenging as it involves designing a dynamic encoder-decoder architecture with multi-head attention layers. Such a model aims to implement an encoder that can selectively focus on different parts of the input data, such as the remaining locations for pickup and dropoffs. It also aims at implementing a decoder that can focus on relevant features of the encoded input representation, such as distance between locations, to help the RL decide which customers to visit next.

Unlike the traditional VRP, the MVRRPP includes additional constraints related to the ride-pooling aspect of the problem. Such constraints include visiting a dropoff location before a pickup location, visiting a visited location or picking up additional customers when the maximum seating capacity of the vehicle is reached. Implementing these constraints is a significant challenge as the RL agent must adhere to such constraints.

Another challenge of this research is finding the appropriate reward function that guides the RL agent during training to find the optimal or near-optimal solutions when solving the MVRRPP. The MVRRPP consists of a multi-objective function used to measure the solution's cost and quality. These objectives include minimising passenger waiting time and travel time and the total distance travelled by the vehicles. Achieving a balance among these objectives is challenging since improving one objective may result in a trade-off with another.

1.4 Aims and Objectives

The aim of this research is to develop an algorithm that uses reinforcement learning to solve the ride-pooling problem and generate high-quality solutions faster than the traditional metaheuristic methods. The algorithm will optimise the allocation of passengers to vehicles, depending on their location and destination, and optimise vehicle routing by minimising the overall waiting time, travel time and total driving distance. In order to reach the stated aim, the following objectives have been set:

1. Implement the Multi-Vehicle Routing for Ride-Pooling Problem (MVRRPP) using Tabu Search as a benchmark algorithm and evaluate its performance in terms of customer's waiting time, travel impact time, distance travelled and time to generate the solution.

2. Model the MVRRPP as a Reinforcement Learning (RL) problem by finding the appropriate representation of states and actions and defining a reward function that balances the trade-off between minimising customers' waiting and travel time, and finally compare RL's performance with Tabu Search.

3. Evaluate the effect of combining a Reinforcement Learning approach with optimisation based on Tabu Search and compare the performance with that of Tabu Search and RL on their own.

1.5 Proposed Solution

To accomplish the objectives stated in Section 1.4, three experiments will be implemented. The first objective will be achieved by implementing a ride-pooling service using tabu search, an algorithm widely used in research to solve the VRP. In this experiment, the algorithm's initial solution will be generated by distributing the customers equally to each driver's route. The initial solution will then be used by the algorithm to iteratively explore and refine the solution space until an optimal solution is found. A solution is considered optimal if it cannot minimise the waiting time, travel time, and distance travelled any further.

The second objective will be achieved by implementing a reinforcement learning algorithm called REINFORCE, a policy gradient algorithm with a baseline model that aims to reduce the variance of the gradient estimates. In this experiment, the MVRRPP will be modelled as an RL problem in which the state will consist of information on the current environment, such as remaining customers, the driver's starting location, and vehicle occupancy and the actions will consist of selecting a location. The reward function will consist of minimising the customer's waiting time, travel time, and the distance travelled by the vehicle while also minimising the number of remaining customers left unpicked at the end of a MVRRPP instance. A dynamic attention model with a dynamic encoder-decoder architecture will be implemented, where the encoder will be used to process the location coordinates and create representation embeddings, and the decoder will be used to generate the output sequence incrementally by selecting locations (actions) at each step based on a probability distribution.

The third objective will be addressed by using the output solution generated from the second experiment as an initial solution for the tabu search algorithm. This experiment aims to determine whether reinforcement learning can improve the solution quality in the tabu search by bootstrapping it with a different initial solution.

Each experiment will evaluate the MVRRPP on various scenarios, encompassing different numbers of customers and drivers. For each scenario containing 4, 5, and 6 drivers, scenarios with 20, 50 and 100 customers will be created. A dataset will be

created for each scenario, containing 500 different MVRRPP instances. Evaluating each experiment in different scenarios helps determine performance on larger problem instances.

1.6 Contributions

While prior research focuses on solving the VRP for logistics purposes, this research aimed to solve the MVRRPP, a variant of the VRP, with the additional complexities related to ride-pooling. This problem required dynamic and real-time decision-making based on continuously changing data. As a customer requests a ride, the algorithm has to dynamically assign a vehicle to that customer and update the vehicle's route to include the new customer's pickup and dropoff location. This research adapted the REINFORCE algorithm with a dynamic attention model to solve the MVRRPP. Using a dynamic encoder-decoder architecture, the model efficiently generated the routes for multiple vehicles serving passengers with similar destinations.

The model's performance was compared with Tabu Search (TS), a metaheuristic algorithm often used to solve the VRP. Evaluating the MVRRPP using TS proved to be effective, achieving the highest quality solutions compared to the RL model. A hybrid approach was also implemented that combines RL with TS to optimise RL's solution further. The solution achieved using the RL model was used as an initial solution for the TS to begin its search process. This approach was only effective in reducing the distance travelled.

The overall results showed that when solving larger MVRRPP instances, the RL model achieved quicker solutions than metaheuristic methods but with a trade-off in solution quality. This demonstrates how the RL model can be applied to real-world ride-pooling scenarios, as it can produce a scalable solution within reasonable execution times.

1.7 Document Structure

The rest of this document is structured as follows:

Chapter 2: Background

Provides background information on the study's relevant research areas. It will include an overview of the vehicle routing problem, metaheuristic algorithms, deep-learning architectures such as neural networks, deep learning, graph neural networks, transformers, and reinforcement learning.

Chapter 3: Literature Review

It reviews existing research comprehensively, including solving different variants of the vehicle routing problem using metaheuristic algorithms and reinforcement learning approaches.

Chapter 4: Methodology

Comprises the design and implementation of three experiments. It first describes the formulation of the MVVRP and its cost function, followed by a detailed description of the techniques used to implement tabu search, reinforcement learning, and the hybrid approach that uses tabu search with reinforcement learning.

Chapter 5: Results & Evaluation

Evaluates the results of the experiments outlined in the methodology chapter. The performance of each experiment was evaluated using the solution running time and the cost function, which consisted of waiting time, travel time, and distance.

Chapter 6: Conclusion

This chapter revisits the aim and objectives and discusses what was achieved. A summary of the findings obtained in the results chapter is given, followed by possible future work and, some the final remarks.

2 Background

This chapter provides the essential background information to understand the problem domain and presents a theoretical framework for the discussed techniques. The research covered in this chapter includes background information on the vehicle routing problem and its variants, metaheuristic techniques, and background information on deep-learning architectures such as neural networks, deep learning, graph neural networks, transformers, and reinforcement learning.

2.1 The Vehicle Routing Problem

The VRP aims to find efficient routes for vehicles fleets that provides a service within a specified duration to a group of customers. Services include delivering goods and transportation systems [9]. The first introduction of the VRP was by Dantzig and Ramser [10], where they formulated the problem as the 'Truck Dispatching Problem'. This problem consisted of finding the shortest routes for a group of gasoline delivery trucks from a central terminal to different service stations while satisfying oil demands and minimizing the total distance travelled by the delivery trucks. Using a linear programming formulation, they propose the first mathematical formula for determining a nearly optimal solution.

Recent VRP models have evolved significantly from the original formulation of the problem to cater for real-world scenarios, such as time windows for delivery and pickup, traffic conditions affecting travel time durations, and demand which changes dynamically through time [11].

2.1.1 Problem Notations

The VRP can be defined as a graph in which the vertices represent the different locations and road junctions while the arcs represent the roads. The graphs and their corresponding arcs can either be directed or undirected, depending on whether they allow movement in only one direction (such as one-way streets) or in both directions. Every arc has a cost, which contains its distance and duration. The type of vehicle and timeframe when the arc is traversed may influence the cost [9].

When defining the VRP on an undirected graph, $G = (V, A)$, where $V = \{0, 1, ..., n\}$ represents the vertex set and $A = \{\{i, j\} : i, j \in V, i \neq j\}$ represents the arc sets. One of the vertices (Vertex 0) represents the depot, which includes m identical vehicles, each at capacity Q. A non-negative demand $q_i \leq Q$ is allocated to each customer $i \in V \setminus \{0\}$ (excluding the depot) [6].

The solution to the VRP can be defined as the total costs linked to the arcs, where each arc represents a vehicle route. The objective of the VRP is to find the set of routes that minimise the costs. Each route is assigned to a single vehicle and contains the sequence of vertices the vehicle visits [9]. Each vehicle must start and finish each route at the depot and not exceed its maximum capacity, Q. Each customer location must be visited by one vehicle, where the customer's demands are satisfied in that visit and not split across multiple visits by that vehicle [12].

2.1.2 Extensions and Variants

Different VRP variants were introduced in research to cater for the requirements and constraints in real-life routing problems. Examples of these constraints include the chosen routes, duration, and length. Each variant's objective is to find the optimal solution to the problem while minimising overall costs [13]. Some of the most researched VRP variants are listed below.

The VRP with time windows (VRPTW) involves the addition of time window restrictions associated with each customer. The vehicle must serve a customer i during the specified time interval $[e_i, l_i]$, where e_i represents the earlier arrival time whilst l_i represents the latest arrival time. The vehicle can only visit the customer i after the earliest arrival time e_i. The VRPTW also includes a service time that represents the time the vehicle takes to serve a customer [14].

The VRP with simultaneous pickup and delivery (VRPSPD) is another variant of the VRP, which allows vehicles to perform concurrent pickups and deliveries during a route. Each vehicle starts by loading the items for delivery at the depot and finishes by returning any collected items to the depot. When visiting a customer, the vehicle can simultaneously carry out delivery and pickup demands while ensuring it is within its load capacity when picking additional items. The VRPSPD aims to establish a series of routes that uses a limited amount of vehicles to satisfy the customer's demand while staying within a maximum travel distance [15].

The Multiple Depots VRP (MDVRP) involves several depots from which vehicles can depart and serve customers. The MDVRP requires that one vehicle visits each customer and that each vehicle begins and finishes its route from the same depot. The first stage of MDVRP consists of determining which depot to assign to a group of customers based on their distance to that depot. In the second stage, customers from the same depot are allocated to multiple routes while ensuring the vehicle is within its maximum capacity. The last stage involves determining each route's sequence for the vehicle to serve the customers. Similarly to other variants, the MDVRP objective is to reduce the overall delivery time (by minimising the number of routes) and operation costs (by minimising the number of vehicles used) [16].

The Electric Vehicle Routing Problem (EVRP) extends on the VRP to include electric vehicles (EVs) and operations such as recharging and minimising the total energy consumption [17]. Due to EVs' battery constraints, EVs have a shorter driving range than vehicles with an internal combustion engine, which restricts their overall driving distance. When planning longer routes, the charging of EVs at a charging station must be accounted for [18]. The EVRP aims to find the most efficient routes that minimise the EVs' energy consumption while considering constraints such as the battery capacity and limited availability of charging stations (CSs) within the area. If their battery energy gets depleted, EVs can visit CSs en route, including the depot where the EV battery can get recharged. Several factors, such as the speed, travel distance and overall vehicle load, may affect the EV's battery consumption [19]. Companies such as those in the logistics distribution sector are utilising EVs in their day-to-day operations to reduce their carbon footprint. The EVRP enables them to determine the most efficient routes for a fleet of EVs to serve a collection of customers while evaluating the constraints associated with electric vehicles [20].

2.1.3 Multi-Vehicle Routing for Ride-Pooling Problem

This research investigates the MVRRPP. This problem involves finding the most efficient routes for driver vehicles to pick up and drop off a set of passengers while minimising costs. Each driver and customer are located in different geographical locations. Each passenger has a pickup location and a drop-off location. Vehicles must first visit passengers' pickup locations before visiting their corresponding drop-off locations. When picking up customers, a driver must start and end a route from the same location and should remain within its seating capacity. The objective of the problem is to minimise the total distance travelled by vehicles while reducing passengers' waiting time and travel time. The waiting time consists of the time taken for the vehicle to pick up a passenger, whilst the travel time consists of the extra time a passenger spends inside the vehicle between pickup and dropoff due to ride-sharing, where the driver may pick up or drop off other customers.

2.2 Metaheuristic Algorithms

Metaheuristics are problem-solving techniques that operate on a high level to tackle various optimisation problems [21]. Optimisation problems, such as those that are NP-Hard, are too complex to search for every possible solution, making it impossible to find the exact optimal solution. Metaheuristic algorithms aim to efficiently find acceptable solutions for optimisation problems in a feasible time [22].

Metaheuristic algorithms balance local exploration and global search, leveraging randomization to achieve diverse solutions. Randomization enables metaheuristic algorithms to explore beyond local search boundaries and search a broader, global scale of potential solutions, making them well-suited for global optimisation and nonlinear modelling. The core elements of every metaheuristic algorithm are exploitation (intensification) and exploration (diversification). Exploration involves creating different solutions to explore a broader search space on a global scale, whilst exploitation involves using the information obtained from a valid solution in the local region to explore search within that specific region. A well-balanced combination of exploitation and exploration typically enhances the likelihood of selecting the best solutions that converge to optimality and attaining the global solution [23].

2.2.1 Metaheuristics Techniques

Metaheuristic algorithms fall into two primary groups: single solution-based metaheuristics and population solution-based metaheuristics. Single solution-based metaheuristics begin the optimisation process using a single solution that gets modified and refined throughout each iterative step. These algorithms could, however, get trapped at local optima and might not extensively explore the entire search space. In contrast, population-based metaheuristics start by creating a set (population) of solutions, which are then updated iteratively. By leveraging multiple solutions, these algorithms could explore more regions of the search space and increase the likelihood of escaping a local optimum. Metaheuristic algorithms can further be grouped into four categories based on their behaviour: swarm intelligence-based, evolution-based, physics-based and human-related algorithms. Swarm intelligence algorithms derive from the social behaviours of animals, insects, birds, or fish. Such algorithms include Particle Swarm Optimisation (PSO) and Ant Colony optimisation. Evolution-based algorithms derive from natural evolution, where a population of solutions are randomly generated at the start, and the best solutions are selected to create new individuals. Such algorithms include the Genetic algorithm and Tabu Search. Physics-based algorithms derive inspiration from the principles governing the universe's physics, such as Simulated Annealing, whilst human behaviour-related algorithms derive inspiration from human behaviour, such as the Teaching learning-based optimisation algorithm (TLBO) [24].

2.2.2 Types of Metaheuristic Algorithms

Tabu Search

Tabu Search (TS) is a local search metaheuristic algorithm that aims to solve optimisation problems by utilising exploration strategies and adaptable memory approaches that help it avoid local optima [25]. TS uses a tabu list to store past selected solutions, enabling the search to avoid reselecting the same solutions and explore alternative solutions during the search process [22]. TS may employ different types of memory structures. Such memory structures include short-term, which involves storing the most recent list of solutions; intermediate-term, which implements intensification rules to focus the search on promising regions within the search space; and long-term, which focuses on diversification rules to guide the exploration towards new regions of the search space. In TS, the algorithm starts from an initial solution and generates neighbour solutions from that initial solution. It then uses the objective function to find the best solution from the set of neighbours [25]. The neighbour solutions are generated by applying a move mechanism, such as swapping and replacing elements within the current solution to create other potential solutions for the neighbourhood [26].

Simulated Annealing

Simulated Annealing (SA) is a stochastic local search method for solving combinatorial optimisation problems. SA draws significant inspiration from analogising the physical process of annealing in solids and solving combinatorial optimisation problems. Annealing in condensed matter physics represents the thermal technique used to acquire low-energy states of a solid within a heat bath [27]. Using this analogy for SA, the energy state represents the SA's objective function, and the shift in the energy state represents the transition from the current solution to a neighbouring one [23]. SA accepts one of two types of solutions in each iteration: an improved solution and a non-improving solution. By accepting 'hill climbing' moves (worse solutions), SA avoids local optima when searching for a global optimum. The algorithm's temperature parameter determines the probability that the algorithm accepts these non-improved solutions. The 'hill climbing' moves occur less often as the temperature parameter decreases, causing the probability of accepting worse solutions to decrease, allowing the algorithm to converge towards better solutions. The algorithm may still allow for occasional exploration of less optimal solutions to avoid getting trapped in the local optima [28].

Genetic Algorithms

Genetic Algorithms (GA) are stochastic search techniques inspired by genetics and natural selection principles. GA encode the search problem's decision variables into finite-length strings that represent the candidate solutions and are composed of specific characters. These strings are called chromosomes, while the specific characters are called genes, and their values are called alleles. These methods have been applied to the Traveling Salesman Problems (TSP), where, in this case, the routes are the chromosomes, the city is the gene, and the allele is the value representing the specific city [27].

GA begin the search process by randomly creating an initial population of candidate solutions. The objective function computes the fitness of these solutions to prioritise the individuals better suited for reproducing and propagating the next generation (iteration). The individuals (parents) chosen to reproduce the next generation of children are combined through a process known as crossover [23]. The crossover aims to create chromosomes (children) that could outperform both parents by inheriting the best traits from each [29]. A new generation with improved fitness is formed by removing the parents and replacing them with the children. Mutation is periodically incorporated into the population to avoid reaching a local optimum and to encourage exploration within the parameter space. The greater the number of iterations in GA, the higher the likelihood of it converging to a population containing identical members where the optimal solution would have been reached [23].

2.3 Deep-Learning Architectures

2.3.1 Neural Networks

Neural networks are computational systems that draw inspiration from the structure of the brain's neurons and their interconnections. Neural networks consist of neurons organized into a structured sequence of at least three layers: the first layer consists of the input layer, the middle layer consists of the hidden layers, and the last layer is the output layer. This neural network's structure, organized layer by layer, is commonly known as a feed-forward network. During a process called forward propagation, data is sequentially transmitted from the input layer and passed through each hidden layer before reaching the output layer. The connections between nodes are called weights, consisting of numerical values that may be positive or negative [30]. Figure 2.1 illustrates a multilayer feed-forward neural network, where each neuron, represented through a circle, is connected to all the other neurons in its neighbouring layers.

An activation function is applied during forward propagation, which uses the

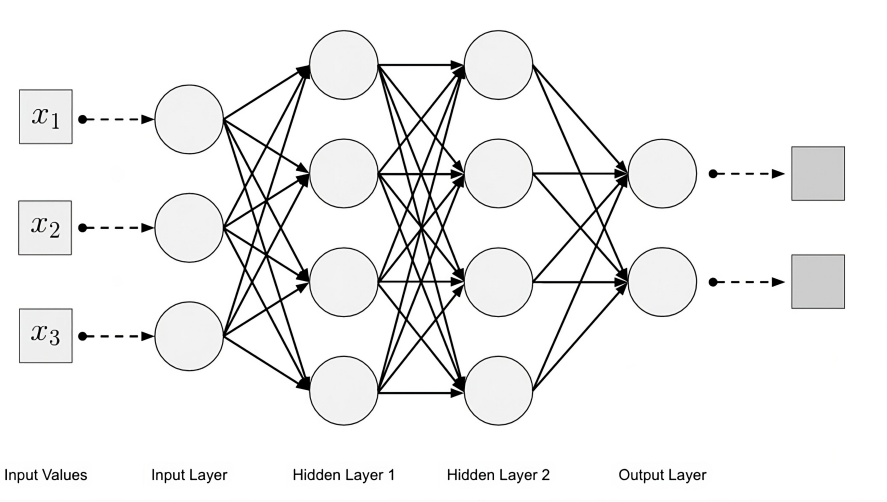


Figure 2.1 Multilayer Feed-Forward Neural Network (Extracted from: Patterson and Gibson [31])

inputs, weights, and biases to calculate the neuron's output value. This output value is then used as input for the next layer. Activation functions determine whether neurons are 'activated'. If the output of the activation function exceeds a certain threshold, typically zero, the neuron is activated [31]. The activation function can be represented using the following equation [32]:

$$f\left(b + \left(\sum_{i=1}^{n} w_i \; x_i\right)\right) \tag{2.1}$$

where $f()$ is the activation function such as tanh or ReLU. Training a feed-forward neural network involves utilizing a learning algorithm such as backpropagation. The backpropagation algorithm is used to estimate the gradient of the loss with respect to the model's parameters (weights). An optimisation algorithm then uses these gradient estimates to compute parameter updates [31].

2.3.2 Deep Learning

Deep learning is a subcategory in machine learning that uses neural networks to enable computational models constructed with multiple layers of processing to acquire knowledge of representations in data with varying levels of abstraction. During deep learning, multiple levels of representations are created by combining non-linear components. Each component is responsible for converting the representation at its current level, beginning with the initial raw input, into a more abstract representation at the next higher level [33]. The initial layers, situated near the data input, acquire knowledge of the basic features, whereas the upper layers acquire knowledge of the more complex features using the data from the lower-level features. This structure

provides a detailed hierarchy of the representation of features [34].

Deep learning uses the backpropagation algorithm to identify complex patterns within large datasets by instructing the network on which parameters to modify. These parameters are responsible for computing each layer's representation based on the preceding layer's representation. Deep learning can also detect complex patterns within data in high-dimensional spaces. This has led to significant breakthroughs in addressing challenges that have long been difficult for other machine learning algorithms [33]. Such breakthroughs include areas in computer vision, natural language processing, speech recognition, and science areas such as medicine [32].

2.3.3 Transformers

Vaswani et al. [35] introduced the Transformer, a network architecture that relies on attention mechanisms for computing input and output representations, eliminating the need for recurrent and convolutional layers. The Transformer uses a stacked self-attention and point-wise with fully connected layers in both the encoder and decoder. The encoder consists of N identical layers, each consisting of a sub-layer comprising a multi-head self-attention mechanism and a fully connected feed-forward network. Each layer has a residual connection with layer normalisation. Similar to the encoder, the decoder also includes N identical layers with an additional sub-layer responsible for performing multi-head attention on the encoder's output stack. The decoder also includes masking on the self-attention sub-layer, restricting positions from attending to subsequent positions in the sequence.

An attention function maps a query vector and a collection of key-value pair vectors to generate an output. It is calculated as a weighted sum that uses the query's compatibility function with its corresponding key to allocate the weight to each value. Transformers utilise scaled dot-product attention for the attention function, which calculates the dot product between the query and all the keys, which is then divided by $\sqrt{d_k}$ and then applied to a softmax function, which determines the weights assigned to the values. The outputs are calculated as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.2}$$

where matrix Q represents the set of queries, matrix K represent the keys and V represent the values. The Multi-head attention layer uses multiple attention layers (heads) that allow the model to focus on different aspects of the sequence in parallel. Each attention head attends to different learned representation subspaces.

2.3.4 Graph Neural Networks

Graph data, which consists of a set of vertices and edges, is often used to depict the network structure of interconnected data. The vertices represent the entities in graph data, while the edges depict the relationship between these entities [36]. While traditional deep learning algorithms have accomplished significant achievements in identifying complex patterns in Euclidean data (like images), several other applications require a graph-based structure to represent data more effectively. An example is e-commerce, where a graph-based learning system can utilise user-product interactions to generate potential recommendations. Graphs are also used in chemistry to model molecules as graphs and identify new drugs based on their bioactivity [37].

Graph learning involves mapping graph features into feature vectors while keeping the exact dimensions within the embedding space. Without reducing the graph's data to a lower dimension, a graph learning model can transform the graph data into the output of a graph learning architecture while identifying the complex relationships between vertices. Many graph learning approaches extend deep learning methods to represent graph data as vectors [36]. An example of a deep learning architecture that works with graph data is Graph Neural Networks (GNN). The GNN's architecture involves passing the input graph to the hidden nodes to acquire knowledge of the representations in the graph data [38].

GNNs can generate outputs for various graph analysis tasks using the graph structure and node content details as input. The GNN can generate outputs at node, edge, and graph levels. The output generated at a node level corresponds to tasks involving node regression and classification, such as ConvGNNs and RecGNNs [37]. A GNN updates the node representations by combining their representation from the prior iteration with the representations of neighbouring nodes [39]. The representations are established by the relationships between vertices using the shared weighted connections [38]. Once the GNN learns the representations of the nodes, the GNN aims to categorise nodes into predetermined classes through node classification [39]. Output tasks at the edge level involve link prediction, which consists of anticipating the presence of an edge between two specified nodes and edge classification, which involves categorising edge types [40]. On a graph level, the output tasks involve graph classification, where the GNN carries out readout and pooling operations to learn graph representations [37].

A Graph Attention Network (GAT) is a GNN architecture that utilises an attention mechanism (inspired by natural language processing) for learning graph representations [41]. By concentrating on the most relevant inputs, attention mechanisms can create a representation for a singular sequence of varying sizes using a self-attention strategy. Using this strategy, the attention-based architecture in the

GATs can calculate the hidden representations of every node. The masked self-attentional stacked layers architecture allows nodes to attend to their neighbourhoods' features by assigning different weights among different nodes within the same neighbourhood without requiring prior knowledge of the graph structure [42].

Attention-based GNNs such as GAT tend to encounter issues with over-smoothing and overfitting. Overfitting may occur with the learned attention functions due to the masked self-attention mechanism restricting the computation of attention weights for only direct neighbours. Over-smoothing may occur when the connected nodes on opposing sides of the class boundary exchange information through the edges. When the multiple attention layers are stacked, these node features may become overly smoothed, making them indistinguishable [41].

2.4 Reinforcement Learning

RL is a machine learning technique where an agent utilises trial and error to learn how to interact in an environment and achieve a particular objective. The agent receives a positive or negative reward based on interactions with the environment. The agent considers the reward as feedback for the actions taken and uses it to further its knowledge of the environment and improve its decision-making. The agent can decide whether to take actions based on existing knowledge of the environment or experiment with actions it has never attempted in that particular scenario [43]. RL aims to find the optimal policy function that provides the best action for any given state. The agent determines the optimal policy through repeated actions within the environment and maximising the cumulative future rewards [44].

Figure 2.2 depicts the RL agent's interaction with the environment. The agent (controlled by the algorithm) at time step t receives from the environment, state s_t. The agent then interacts with the environment by carrying out action a_t, which results in the environment transitioning to a new state, s_{t+1} in the following time step. The state consists of the essential information from the environment for the agent to take the optimal action. When transitioning to the following state, the environment sends feedback to the agent as a scalar reward r_{t+1}. The agent aims to discover a policy π that maximises the cumulative discounted reward. The agent can improve its policy by using knowledge from each state transition $(s_t, a_t, s_{t+1}, r_{t+1})$ [45].

2.4.1 Markov Decision Processes

In RL, the relationship between the learning agent and the environment is defined using the Markov Decision Processes (MDP). We establish this relationship using actions, states and rewards. The MDP represents a conventional approach to

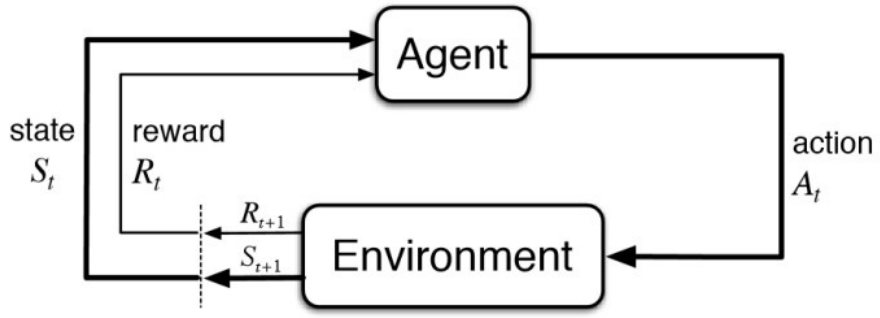


Figure 2.2 The RL's agent interaction within the environment. (Extracted from: Sutton and Barto [46])

formalising sequential decision-making problems. The actions taken in these problems impact not only the current rewards received but also the future states, which in turn affect future rewards [46].

A finite MDP consists of a set of states S, a set of actions A, and a probability distribution $p(s', r\,|s, a)$, where $\{s, s'\} \subseteq S$, $a \in A$, and $r \in \mathbb{R}$, that defines the probability of transiting to state s' from s and obtaining the reward r when applying action a . An MDP can also have the discount factor $\gamma \in [0, 1]$ [46]. The Markov property establishes that the transition function is dependent on information from the current state, selected action, and the resulting state and is independent of past actions and states. Elements like the rules and physics of the environment are stationary and remain consistent. An example is chess, where the player does not need to recall past moves to play his next move [43].

2.4.2 Policy and Value function

At each time step t, the agent obtains states $s_t \in \mathcal{S}$ from the environment, where it then chooses the actions $a_t \in \mathcal{A}$ based on the policy $\pi(a_t|s_t)$. The policy maps the states s_t with the actions a_t. When the agent performs an action a_t, it receives a reward $r_t \in \mathcal{R}$ from the environment. In each state s_t, the agent aims to maximise the cumulative discounted reward $G_t = \sum_{k=0}^{\infty} \gamma^k r_{k+t}$ [47].

The value function $V(s)$ determines how beneficial it is for the agent to be in a particular state while following a given policy π by estimating the value of G_t corresponding to other states. Different policies can provide different values for identical states [48]. This state-value function relies on the specific policy π that the agent adheres to [49]:

$$V^{\pi}(s_t) = \mathbb{E}\left[R_t | s_t = s\right] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right] \tag{2.3}$$

The action-value function Q can be described as the valuation of the outcome when action a is chosen within state s, and then the agent starts following policy π:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}\left[R_t | s_t = s, a_t = a\right] \tag{2.4}$$

$$= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right] \tag{2.5}$$

The optimal policy π^* refers to a policy that acquires the highest cumulative reward over an extended period $\pi^* = \underset{\pi}{\text{argmax}}\, V^{\pi}(s)$. Derived from the Bellman optimality equation [50], Equation (2.6) represents the optimal policy using the state-value function while Equation (2.7) represents the optimal policy using the action-value function.

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{2.6}$$

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a) \tag{2.7}$$

2.4.3 Reinforcement Learning Algorithms

RL algorithms can be categorised as model-free and model-based methods. Model-free methods use trial and error to expand their knowledge of the environment and update the policy accordingly to achieve optimal rewards. Since model-free methods lack information about the transition model and reward function, they need to interact with the environment repeatedly to understand the system's behaviour. Examples of model-free RL are Temporal Difference (TD) and Monte Carlo (MC). In model-based methods, the agent does not depend on experimentation, unlike model-free methods, but instead utilises a predefined learned model to forecast various states and their corresponding rewards. An example of a Model-based RL algorithm is dynamic programming [43].

RL can also be categorised as value-based or policy-based. In value-based RL, the value function is calculated for each state and used to evaluate the policy [48]. In policy-based RL, the policy is updated and optimised directly through each iteration

until it maximises the cumulative return [51]. Actor-critic algorithms combine both value-based and policy-based methods. The critic computes the value function, and the actor adjusts the policy based on the values of the critic [48].

Dynamic Programming (DP) consists of algorithms that use a model of the environment (represented as an MDP) to find an optimal policy [46]. Since DP algorithms are model-based, they require both the transition model and the reward function. Applying DP requires the problem to include the following properties: optimal substructure and overlapping sub-problems. The optimal substructure refers to the property where the best solution for the problem can be deconstructed into solutions for its smaller sub-problems. The overlapping sub-problem property is when the finite sub-problems are encountered recursively, allowing their respective solutions to be stored for potential reuse [51]. Two widely utilised techniques in DP are value iteration and policy iteration. Policy iteration aims to find the optimal policy by iterating through different policies and retaining only the policy that delivers the most significant overall rewards. The optimal policy is determined when the cumulative rewards of π converge. In contrast, instead of evaluating multiple policies, value iteration determines the optimal policy by finding the optimal value functions and sequentially navigating through each state to determine the actions associated with the highest values [52].

MC methods are model-free RL, which rely on learning through the experience they acquire when interacting within an environment (without prior knowledge). Unlike DP, which requires all probability distributions of the transitions, MC methods only require a sample of states, actions and their rewards. MC methods can estimate the state-value function for a given policy π, where the expected returns from the policy are averaged when visiting a state multiple times until the average result converges to the expected value [46]. To estimate the state-value function $V_\pi(s)$ when following policy π, we first gather a set of episodes traversing through state s. Each occurrence of state s in an episode is called a 'visit'. MC methods can then perform the estimations using the first-visit MC or every-visit MC method. The first-visit MC calculates the average estimation in the episode by using only the returns of the first visit to state s. In contrast, the every-visit MC method calculates the average estimation in the episode using all visits to state s [51].

TD is another model-less RL consisting of a combination of concepts from DP and MC. Like MC, TD learns from the experience it acquires when interacting with the environment without requiring an existing model of the environment. When compared to MC, TD allows finding the return after the first time step, while MD requires completing the episode to obtain the return. Another distinction with MC is that when carrying out experimental actions, MC tends to discount or ignore episodes, which can hinder learning; in contrast, TD extracts knowledge from every transition, regardless of their following actions, making them less prone to such challenges [46].

Similarly to DP, TD uses bootstrapping, which involves using the previously calculated value functions to estimate the current value function. Two widely used TD methods are SARSA and Q-learning. Q-learning is an off-policy algorithm where the agent follows an equal probability policy. This policy ensures that all available actions in all states have an equal chance of being selected during training. This approach allows the agent to explore the environment more thoroughly and change to an optimal policy. In contrast, SARSA uses an on-policy algorithm where the policy it follows during training (behaviour policy) is the same as the policy it aims to discover, the target policy. This target policy is assumed to be the optimal policy. The agent is considered on-policy if the target and behaviour policies are the same [52].

2.4.4 Policy-based Methods

Policy-based methods involve learning a parameterised policy that directly selects actions without utilising a value function. Some policy-based approaches may employ a value function to train the policy parameters but do not use it to determine actions [44]. In contrast to value-based RL, a policy-based approach offers improved convergence, simpler policy parameterisation and is better suited when working with continuous or high-dimensional action spaces [51].

Policy-based methods use gradient-free or gradient-based optimisation techniques to update the optimal policy. Gradient-free techniques use heuristic search to find better policies. Such methods consist of evolution strategies, such as hill climbing within a subset of policies. Although gradient-free techniques can efficiently explore low-dimensional parameter spaces, they might not scale efficiently to larger, more complex models. In gradient-based methods, the gradients provide valuable feedback on how to enhance a parameterised policy. In policy gradient RL, deterministic and stochastic approximations are used to find the average of the plausible trajectories (generated by the current policy parameterisation) and then calculate the expected return. Despite their computational demands, gradient-based methods remain the preferred choice due to their efficiency when working with policies with many parameters [45].

REINFORCE is a policy gradient algorithm that uses a single trajectory or a set-sized mini-batch of trajectories with a gradient estimator to calculate a stochastic estimated gradient [53]. REINFORCE stands for **RE**ward **I**ncrement = **N**onnegative **F**actor x **O**ffset **R**einforcement x **C**haracteristic **E**ligibility [54]. The REINFORCE algorithm utilises Monte Carlo techniques to sample episodes and approximate the gradient of the cumulative reward. While providing unbiased estimations, these estimations may experience high variance, increase sample complexity, and impact the learning rate [55]. Without altering the bias, the variance in these estimations may be

reduced by implementing a baseline to the REINFORCE algorithm. The bias is maintained by subtracting the baseline value from the expected return [54]. The REINFORCE algorithm can adapt to different neural network architectures that provide arbitrary outputs, allowing it to be applied to various problem domains. It also offers the flexibility to integrate with other gradient methods, such as backpropagation [56].

2.4.5 Deep Reinforcement Learning

In contrast to traditional RL, which uses analytical approaches to determine the function approximation, Deep Reinforcement Learning (DRL) utilizes deep neural networks, allowing it to work with high dimensional problems [51]. DRL uses deep learning techniques to extract observation data from the environment. This observation data helps in understanding the current state of the environment. RL then determines the appropriate action based on the current state and evaluates the expected return of the action taken in the current state. The action and state spaces in traditional RL algorithms contain low dimensions and, therefore, are normally represented as tables or arrays for the approximate value functions. Real-world implementations, however, contain continuous and high-dimensional action and state spaces that cannot be represented as tables (curse of high dimensionality), requiring the approximate value functions to be represented as parameterised functions with weight vectors. This can be achieved using DRL, where the model features, policies and value functions are obtained using the function approximation and generalised to construct an approximation of the complete function through the deep neural networks. This, therefore, allows DRL to generalise states it didn't see during training and scale to large (potentially infinite) state spaces [49].

Deep Reinforcement Learning Algorithms

Deep Q-Network

Deep Q-Network (DQN) is a value-based DRL algorithm created by Mnih et al. [57] that combines Q-learning with deep neural networks to learn policies from high-dimensional sensory inputs. DQN contains a modified Q-learning implementation that uses an experience replay to store the agent's experiences in a replay memory at every time step. This replay memory is then accessed to execute weight updates [46]. These transitions are stored in memory as follows: $(s_t, a_t, s_{t+1}, r_{t+1})$. The RL agent then samples the transitions and trains from past experiences. An experience replay can reduce total interactions with the environment and variance by sampling batches of experience. To improve stability during training, the DQN uses a target network that updates its weights after a set number of steps to match the policy network's weights.

Using a fixed target network allows the DQN to avoid calculating the fluctuating TD errors stemming from its rapidly changing Q-value estimates [45]. The Double DQN algorithm, developed by Hasselt et al. [58], aims to reduce the overestimation problem in DQN when estimating the values of the actions. Double DQN does this by decoupling the action selection from the action evaluation (target network), improving value accuracy and policy quality.

Proximal Policy Optimisation

Proximal Policy Optimisation (PPO) is a policy gradient DRL algorithm that optimises the policy by alternating between collecting data from the policy through interactions with the environment and performing multiple optimisation steps using the collected data through stochastic gradient ascent. PPO achieves the same level of performance and efficiency of data as Trust Region Policy Optimisation (TRPO), a policy gradient algorithm which aims to optimise non-linear policies. Unlike TRPO, PPO uses only the first-order optimisation algorithm to achieve these results. When analysed on benchmark experiments such as playing Atari games and simulated robot locomotion, PPO performed better than current state-of-the-art approaches such as Advantage Actor-Critic (A2C) and performed similarly to Actor-Critic with Experience Replay (ACER) [59].

3 Literature Review

This chapter reviews existing literature on metaheuristic and RL algorithms used to solve different variants of the VRP. The first section of this chapter includes research that employs metaheuristic algorithms, such as Tabu Search, Genetic Algorithms and Simulated Annealing, whilst the second section includes research involving RL algorithms, such as DQN, A2C and REINFORCE.

3.1 Metaheuristic Algorithms for Vehicle Routing Problems

Li and Li [1] propose using an improved TS algorithm to solve the VRP with soft time windows and stochastic travel and service time (SVRP-STW). This variant of the VRP consists of a depot with a fleet of homogeneous vehicles that must find the optimal path to deliver goods to customers within a time window. In this problem, however, the vehicle can visit the customer outside the time window but will incur costs. The objective function consists of the penalty costs for arriving before or after the customer's time window and the total vehicle's travel cost. To solve the SVRP-STW, they propose using an improved TS algorithm. For the initial solution, containing the vehicles' routes, they implement a greedy algorithm that considers the capacity, time window and travel distance constraints when creating the initial routes. Throughout each iteration, customers close to the current node and with a limited time window are preferred and inserted into the current route at the most suitable feasible location. A new route is created when a vehicle reaches its capacity limit and cannot accept new customers. Their improved TS algorithm includes an adaptive tabu list that adjusts the tabu length interval based on the quality of solutions found and an adaptive neighbourhood structure that explores other neighbourhoods once a neighbourhood is explored. Compared to other metaheuristic algorithms, such as SA and GA, on 25, 50 and 100 customers, their proposed algorithm achieved impressive results in terms of execution time and solution quality, highlighting the effectiveness of this algorithm for solving SVRP-STW problems.

Gmira et al. [60] also propose using TS to solve a variant of VRP known as the Time-Dependent VRP with Time Windows on a Road Network $(TDVRPTW_{RN})$. This VRP variant aims to find the shortest path (in terms of time) for customers by considering the travel durations throughout the day. Similarly to Li and Li [1], they also generate the initial solution using the greedy algorithm; however, for the neighbourhood solution, they utilise the CROSS exchange algorithm [61] that swaps customers in a route without allowing reverse segments. This is suited for scenarios involving time-window constraints. To determine the feasibility of CROSS's exchange,

they use Dijkstra's algorithm to calculate the shortest distance between each pair of customers at a given departure time. In TS, they implement diversification to encourage the exploration of different regions within the solution space. They do this by employing a different objective over a set number of iterations, such as minimising the total distance rather than the duration. The original duration objective is then restored until the search process becomes trapped again in a local optimum. Their model successfully produced high-quality solutions in problem scenarios containing 200 nodes in 580 arcs within efficient execution times.

Rabbouch et al. [2] explored using GA to solve the Multi-Depot Heterogeneous VRP with Time Windows (MDHVRPTW). This problem involves finding minimal-distance routes for a heterogeneous fleet of vehicles situated in various depot locations to serve customers in different areas while adapting to their different time constraints and demands. They create the algorithm's initial population by assigning customers to the nearest depots and grouping them into clusters based on the number of depots. The algorithm encodes the chromosomes (solutions) as route sequences, encoding customers using their index in the order in which they will be visited. For the mutation and crossover, two parents are selected using tournament selection, which involves selecting the best two chromosomes depending on their fitness function. An offspring is created by randomly selecting a route from the first parent and replacing it with a route from the second parent. A customer is removed from the old route if it is already included in the new route. If a customer remains without a route, they are either inserted into the route in the appropriate position or added to a new route if it does not follow the time window constraints. Their proposed algorithm produced longer routes when compared to state-of-the-art algorithms, such as Variable Neighborhood Search (VNS) [62] and Unified TS (UTS) [63]. VNS is a metaheuristic algorithm that changes neighbourhood structures during a descent phase and an exploration phase to escape local optima [62] while UTS is an adapted TS that dynamically adjusts parameters to navigate the search space and applies diversification strategies to avoid local optima.[63]. The algorithm proposed by Rabbouch et al. [2], however, achieved impressive results in a shorter time, highlighting its effectiveness compared to VNS and UTS.

Park et al. [15] also explore using GA to solve the VRP while focusing on solving specifically the VRP with simultaneous pickup and delivery (VRPPD). This problem consists of a single depot with homogeneous vehicles that can concurrently pick up and deliver goods to customers using the same vehicle on a single route. In their proposed GA model, the VRPSPD solutions are encoded as chromosomes. Every gene contains a vehicle index value and links to three sets of parameter chromosomes: demand location, quantity, and product's weight variety index. The population consists of solutions, each representing a combination of randomly generated vehicle indices

assigned to nodes for delivery or pickup. The algorithm's fitness function included the total costs to deliver the products and the vehicle's total distance travelled based on the Euclidean distance equation. Selecting the parent chromosomes to create the child chromosomes involved using a roulette wheel selection. This consisted of calculating the fitness function for all solutions in population *S* and sorting them according to the highest fitness level. A number, *F*, is randomly selected between 0 and *S*. A partial sum, *P*, is then calculated using the cumulative sum of fitness values from the highest fitness on the list until *F*. The selected solutions in *P* contribute to creating the child's chromosomes. A crossover operator selects two random parents from the selected solutions and creates two child chromosomes by randomly creating a segment on each parent and replacing the segment from the second parent with the first parent segment. Their results indicated that the proposed model handled real-life scenarios within reasonable execution times.

Yu et al. [64] propose using SA to explore a variant of the VRP with Time Windows that includes the addition of delivering to parcel lockers. This variant is called the VRP with parcel lockers (VRPPL). In VRPPL, customers may have their items delivered to their house, nearest parcel locker, or either option. One vehicle can deliver once to a customer's house, whilst parcel lockers can be visited more than once by the same vehicle or different vehicles. The VRPPL's solution can be represented as a two-dimensional array of customer indices and delivery options. The objective function, which involves reducing the distance travelled, is calculated on the array's second row, consisting of the vehicle's visited nodes. For the initial solution, the customers and their delivery type are arranged in ascending order according to the customer's indices. The nearest neighbourhood algorithm then selects nodes closest to the currently visited node.

In their proposed model, the SA algorithm selects the initial solution as the current solution, $\sigma_{current}$, whilst the initial temperature T is set to T_0. The algorithm then uses the $\sigma_{current}$ to find a new solution, σ_{new}, by selecting one from the following neighbourhood moves on the first row: swap, insertion, and inversion. To determine which move is selected as σ_{new}, a random value r_1 is generated. The objective value for a solution σ is represented as $f(\sigma)$. σ_{new} is accepted as the new $\sigma_{current}$ by comparing $f(\sigma_{new})$ and $f(\sigma_{current})$, whilst σ_{new} is accepted as the new σ_{best} if $f(\sigma_{new}) < f(\sigma_{best})$. The difference between $\sigma_{new} - \sigma_{current}$ is represented as Δ. If σ_{new} has a higher objective value, it may still be accepted as a new $\sigma_{current}$ if $r_2 < e^{-(\Delta/\beta T)}$, where r_2 is a random value generated between 0 and 1, β is Boltzmann constant, $e^{-(\Delta/\beta T)}$ is the new neighbourhood solution probability. After an iteration is completed, a constant α is multiplied by T, reducing the temperature. Their findings showed that the SA performed better in small and large VRPPL instances than the Gurobi solver.

To solve the routing optimisation of ride-sharing taxis, Lin et al. [65] suggest

using SA to minimise costs and maximise customer satisfaction. The metrics used to evaluate these objectives included measuring the waiting time, extra riding time (time incurred due to ride-sharing) and travelling milage. In their proposed model, each vehicle travels at a constant speed, where the pick-up time windows, locations of the origin and destinations are known beforehand. Their results indicated that when serving 29 customers, the regular taxi system, which consisted of each taxi serving one customer at a time, would require 29 taxis and a total travel mileage of 375km to serve customers' demands. When repeating the same experiment using their SA model, only 10 taxis were used with a total travel miles of 302km. This indicated that their SA model saved 19% mileage and 66% of taxis remained available, indicating that fewer taxis were needed to serve customers.

Herbawi and Weber [66] also propose using metaheuristics to solve ride-sharing problems. Their research uses GA with an insertion heuristic to solve the ride-matching problem with time windows (RMPTW) in a dynamic ride-sharing environment. Similar to Lin et al. [65], their problem involves drivers and customers with origin and destination locations and a specified time window. Drivers may specify the total distance and travel time and whether they accept detours to accommodate customers. The RMPTW objective is to minimise the total distance and time of vehicles, minimise the time of customer trips, and maximise the total of customers served. The first stage of their model consists of using GA to solve the static rendition of the problem, which consists of the known requests. In the second stage, the model then uses the insertion heuristic to update the GA's solution to accommodate newly received requests. When demonstrated on real-world datasets, their model solved the dynamic ride-matching problem in real-time while providing solutions that balance quality and response time.

3.2 Reinforcement Learning for Vehicle Routing Problems

To improve ride-sharing services, Al-Abbasi et al. [67] present a model-free method that uses DQN to learn optimal dispatch policies that optimally assign vehicles to customers. Their objective was to minimise the supply and demand mismatch, customer waiting time, time users experience due to carpooling, and number of vehicles utilised, consequently reducing traffic congestion and fuel consumption. Each vehicle determines the best action using Q-learning by considering nearby vehicles and understanding how the action affects the reward. When evaluating their model on a New York City taxi dataset of 15 million taxi trips, their model outperformed baseline policies such as those not considering ride-sharing. Their model reduced the number of vehicles by 500 and the passengers' waiting time for the same number of requests.

Guo and Xu [68] explore solving the vehicle dispatching and routing problem in

a ride-sharing autonomous mobility-on-demand system using a Convolutional Neural Network and double DQN. By leveraging historical data, they utilize a DRL framework to make vehicle routing decisions that balance operation cost and service quality. Their model also considers the repositioning of idle vehicles where they relocated to regions with high demand. Using dynamic programming, their model then assigns vehicles to customers' requests based on vehicles that yield the highest reward. Their reward function consists of the passenger's quality of service subtracted from the system's operating cost. When experimented on a New York City dataset with 40, 60, and 80 identical vehicles, their model resulted in an increased service rate and a reduction in the vehicle's travelling distance and average waiting time when compared to other algorithms such as the optimal high-capacity vehicle dispatching algorithm with rebalancing proposed by Alonso-Mora et al. [69] and other strategies involving no rebalancing. Waiting time was, however, higher when compared to the strategy, which considered only the quality service of individual customers; nevertheless, their model compensated for this shortfall in terms of service rate and distance.

Vera et al. [3] propose utilising DRL to solve the Capacitated Multi-Vehicle Routing Problem (CMVRP), which consists of finding routes that minimise the total distance travelled by different vehicles (with different capacities) when serving customers' demands. Each agent's policy is represented through deep neural networks (DNNs) and is trained using RL. All agents (vehicles) access information from all other agents, ensuring a consistent environment state across all agents. They formulate the CMVRP as an MDP, where each agent makes decisions by relying solely on the information from the previous state of the environment and does not consider the actions of other agents. They use the A2C algorithm, a policy gradient method that uses the actor network and the critic network as function approximators to train the network. The stochastic policy is defined by parameters characterising its embedding, decoder, and attention mechanisms. The policy improves iteratively by estimating how changes in the policy parameters impact the expected rewards' gradient, thereby guiding the policy towards maximising expected rewards. The attention mechanism gathers information from all the elements in a set of input nodes, which it uses to create the output sequence. An affinity function is used to generate values that measure the relationship between each node and the previous output of the model. Applying the softmax function to these values provides the attention (importance) of each input element at each timestep, helping the model focus on relevant inputs when generating the output. Their proposed model performed better than the Sweep Heuristic [70], an algorithm that constructs routes from multiple depots and uses multiple heuristics to refine the solution, and the Clarke-Wright Savings Heuristic [71], an algorithm that finds efficient routes for multiple vehicles by combining pairs of delivery points into a single route to minimise total travel distance. However, when compared to Google

OR-Tools, it performed better. They stated that while OR-Tools performed better, their model could determine policies for any VRP configuration in faster execution times.

Similarly to Vera et al. [3], Zhao et al. [8] also propose using DRL to solve the VRP. Instead of applying a critic alongside an actor, Zhao et al. [8] use an adaptive critic. The actor is responsible for creating routing policies through an attention mechanism with graph embeddings, whilst the adaptive critic is a combination of self-critic and normal critic, responsible for optimising the actor's parameters during training to increase the convergence rate and improve performance. The adaptive critic calculates the policy gradient method's baseline to minimise variance during training. A routing simulator is used to create VRP instances that simulate real-world scenarios. It also acts as the model's environment, responsible for interacting with the model. After the vehicle chooses the next customer, the simulator provides the new states, rewards the actor and adaptive critic and updates customer masking. Customers who have already been visited or cannot be visited due to capacity or time constraints are masked. Their actor-network architecture consisted of the encoder (to embed the static and dynamic state to a D-dimensional space), the decoder (to decode the vehicle's embedding to a high dimensional state) and the attention layer (to learn the correlation between the customer point and the current state). Their results indicated that their DRL method achieved better results when compared to baseline algorithms such as ant colony optimisation (ACO). During their research, they also investigated using the DRL's output as the initial input for the local search algorithm using Google OR tools. They concluded that DRL could provide robust initial solutions for local search algorithms, leading it to find solutions that are close to optimal.

Basso et al. [7] use a modified Q-Learning algorithm to solve the Dynamic Stochastic Electric VRP (DS-EVRP). This problem involves determining routes for an EV serving random customers' requests throughout the day, starting and ending at a depot. Customer requests can be deterministic if acknowledged before departure or stochastic if received after the EV leaves the depot. The model uses a chance-constrained policy with two layers to learn a policy that minimises the vehicle's energy consumption and battery depletion when carrying out an action in a given state. They use value function approximation to estimate the value of state-action pairs using look-up tables with reduced state representation. They also use a training strategy that implements rollout heuristics to guide the agent in selecting the best actions when in a state. Their research shows that using RL to plan routes and anticipate charging, vehicles may save an average of 4.8% and up to 12% of energy.

To solve the VRP, Kool et al. [72] propose using an attention-based model consisting of an encoder and decoder. They propose training this model using the REINFORCE algorithm with a greedy baseline. Each attention layer in the model includes a multi-head attention and feed-forward sublayer. The encoder uses the

multi-head attention to embed the graph comprising the customer locations, whilst the decoder then uses these embeddings to output the next node.

Mustakhov et al. [5] propose using RL to solve the Stochastic Dynamic VRP (SDVRP). Using MDP to model the SDVRP, they implement the REINFORCE algorithm to learn an adjustable and responsive policy capable of adapting to the dynamic and stochastic demands of the problem. The optimal policy consists of maximizing the number of customers' demand. Contrary to Kool et al. [72], they use a dynamic encoder to encode the graph each time the agent selects an action. The decoder utilizes multi-head attention to decode the graph of the current state. Unlike Kool et al. [72], they take the current embedding of the graph, consisting of served and unserved customer demand at the current time and the vehicle's current location, as input to determine the next vehicle for the following action. They implement a baseline model that adjusts the parameters of the primary model. The baseline model calculates the reward using a greedy strategy that selects actions (nodes) that most likely yield the highest rewards based on their probabilities. On the other hand, the main model samples actions. The parameters of the baseline model are updated if there is a significant difference between the results of the two models.

Zhang et al. [73] propose using GAT with RL (GAT-RL) to solve the Multi-depot VRP with soft time windows (MD-VRPSTW). Their GAT-RL model uses an encoder with a multi-head attention mechanism to acquire and process location information from various depots and customers simultaneously. They integrate a GAT with the encoder to capture spatial and temporal details within the time window network, producing the outputs of the depots and customer embeddings. The decoder then uses the embeddings, the masking for constraints, and the vehicle's current state to generate sequences containing the customers to visit. In their research, they compare two policies: fixed-order policy and full-pair matching policy. The fixed-order policy uses a fixed order to assign vehicles to customers, whilst the full-pair matching policy expands the vehicle's exploration area. The model's parameters were trained using a policy gradient algorithm, REINFORCE, and a greedy rollout baseline. Their results indicated that the model with the fixed-order matching policy converged faster, although the model with the full-pair matching policy achieved better solutions. The GAT-RL model excelled in solution quality and efficiency compared to benchmark algorithms, such as hybrid GA with iterated local search.

Gupta et al. [74] use a combination of deep RL and heuristics to solve the Capacitated VRP with a time window (CVRPTW). They aim to reduce transportation costs by decreasing the number of vehicles in use and the total distance travelled. Similarly to Zhang et al. [73], their DQN model utilises a multi-head attention layer for the graph attention encoder. The model's decoder then takes the encoded state as input and utilises a fully connected network to generate Q-values for each action (each

customer). The Q-values help determine the action to select. Only feasible customer actions are considered while masking those not meeting the feasibility conditions. An insertion heuristic was used to improve the candidate solution obtained using deep RL. This heuristic involved splitting the routes with the least number of customers and then checking each node in these routes for feasibility by inserting them between the nodes in the remaining routes. The feasible node (location) with the least distance increase is selected and inserted to create a new route. Compared to GA, their model obtained close to optimal solutions with fewer vehicles deployed; however, it incurred an increase in distance travelled cost compared to GA.

Zhang et al. [4] also present a multi-head attention mechanism for an encoder-decoder architecture, which uses RL to develop a multi-agent attention model (MAAM). This model can solve the Multi-VRP with soft time windows (MVRPSTW). The encoder creates the depot and customers' embedding and updates these embeddings using multiple attention layers that perform multi-head attention and feed-forward operations. The decoder then uses as input the encoder outputs, the constraint mask matrix, and the context embedding to generate a sequence of customers for multiple vehicles, wherein, for each timestep, one customer is assigned to a vehicle. Like Zhao et al. [8], already visited customers and customers' demands that exceed the remaining vehicle's capacity are masked. Like [5] and [73], REINFORCE was used to train the MAAM with a baseline model that uses a greedy rollout policy to reduce variance and improve training efficiency during the training process. During each epoch, the policy parameters are updated if the difference between MAAM and the baseline model is significant. Their model outperformed classical heuristics such as GA and local search when experimented on 20, 50, 100 and 150 customers with different capacities. Their model provided robustness when handling different customer numbers and vehicle capacities, allowing it to be applicable in realistic scenarios.

Previous studies, such as [3], [8], [5], [73], [74] and [4], use an attention model (AM) to solve the VRP. In AM, the problem is represented as a graph. The AM uses neighbouring graphs to extract node features and create the solution incrementally using the decoder. The decoder uses predictions to generate a probability distribution across nodes, subsequently choosing one node to add to the partial solution based on this distribution. Peng et al. [75] state that the attention model's embeddings, consisting of the node features, remain fixed throughout and do not reflect the changes in the state as the model makes decisions. They explain that as the model creates a partial solution (a route consisting of a sequence of nodes that start and finish at the depot), the remaining unselected nodes can be considered a new instance that differs from the original instance. They, therefore, suggest that node feature embeddings should be updated based on this new instance. Figure 3.1 illustrates the partial solution generated by the model using the original instance and shows the new

instance created using the remaining nodes.

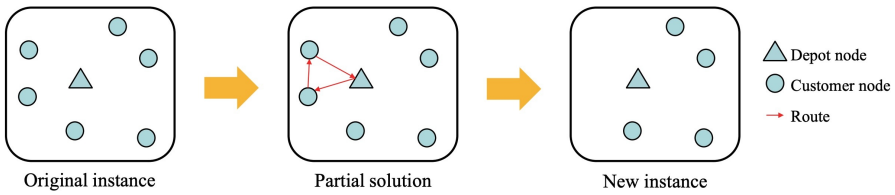


Figure 3.1 The different states of an instance at different steps (Extracted from: Peng et al. [75])

To address this problem, Peng et al. [75] present a dynamic attention model (AM-D) with a dynamic encoder-decoder architecture that updates each node's embeddings during different stages, such as whenever the vehicle returns to the depot. This change allows the model to dynamically discover node features and efficiently leverage hidden structural information across various steps. Unlike the vanilla encoder-decoder AM, the AM-D encoder-decoder updates each node's embeddings, including masking visited nodes when creating the partial solution. They train the AM-D using REINFORCE with a sample and greedy rollout. When experimented on instances with 20, 50 and 100 customers, the AM-D outperforms the AM across all problem instances. This indicates that the AM-D has strong generalization due to the smaller, easier-to-solve partial solutions created at each stage.

3.3 Summary

The research shows that metaheuristic algorithms can effectively produce high-quality solutions for different variants of the VRP. On the other hand, using RL algorithms to solve the VRP has also proven effective, with models able to learn policies for any VRP configuration. RL has also shown that it can serve as an excellent initial solution for local search algorithms. The RL models highlighted in the literature showed how using RL could optimise solutions more efficiently, leading to faster execution times when compared to traditional metaheuristic algorithms. RL's ability to be efficient and robust when handling instances with varying numbers of customers and vehicles highlights its ability to be applied in various real-world scenarios.

Recent research involving RL has incorporated an attention-based model to solve the VRP. This model utilises an encoder and decoder with a multi-head attention mechanism. Peng et al. [75] adapt this model to include a dynamic attention model with a dynamic encoder-decoder architecture, achieving a significant improvement in

terms of performance across different instances when compared to research, which uses the standard attention model to solve the VRP. This research can be further expanded and adapted to solve different variants of the VRP.

4 Methodology

As indicated in Chapter 3, most research focused on solving different variants of the VRP for logistic purposes, where they find the optimal routing strategies to enhance efficiency and reduce cost during transportation operations. Recent research highlights the use of RL together with an attention model. Peng et al. [75] expand upon this research to implement a dynamic attention model for the single depot VRP, which involves picking up items from a depot and delivering them to customers. In this chapter, we will combine the classic metaheuristics algorithms with machine learning techniques implemented by Peng et al. [75] to solve a variant of the VRP known as the MVRRPP. Unlike the problem investigated by Peng et al. [75], this problem adds complexity as it involves using multiple vehicles that must start and finish their route from their respective origin locations, which are different for each vehicle. For each customer, the vehicle must also visit the customer's respective pickup and drop-off location. This problem also requires the vehicles to visit specific locations sequentially, since a customer's pickup location must be visited before their respective dropoff location using the same vehicle.

4.1 Problem Formulation

The MVRRPP can be defined as a pair $X = \langle U, C \rangle$, where U represents a set of origin nodes $U = \{u_0, u_1, ..., u_{nu-1}\}$ with nu indicating the number of origin locations, whilst C represents a set of customer nodes $C = \{c_0, c_1, ..., c_{nc-1}\}$ with nc indicating the number of customers.

Each origin location $u_i \in U$ consists of a pair $\langle x_u, y_u \rangle$, where x_u is the latitude and y_u is the longitude of the origin coordinate. Each customer $c_i \in C$ consists of the pair $\langle p, q \rangle$, where p is the pickup location, q is the drop-off location. Each p consists of a pair $\langle x_p, y_p \rangle$, where x_p is the latitude and y_p is the longitude of the pick-up coordinate. Each q consists of a pair $\langle x_q, y_q \rangle$, where x_q is the latitude and y_q is the longitude of the drop-off coordinate. Each origin location u_i and customer c_i (pickup p_i and dropoff q_i) is two-dimensional and represented in Euclidean space.

A solution consists of a list of waypoints for each driver to visit. Each driver d_i consists of a list of waypoints $d_i = (w_0, w_1, ..., w_n)$, where each waypoint w represents a u_i or c_i (containing either pickup p_i or drop-off q_i). For each w_i containing a p_i , it's corresponding q_i must be visited by the same d_i. In each d_i , the sequence of w_i must first include the p_i before it's corresponding q_i . Each d_i has its respective origin location u_i and therefore must start and end from it. Since the number of drivers nd is equal to the number of origin locations, $nu = nd$.

The problem's final solution π , consisting of a list of drivers, can therefore be represented as $\pi = (d_0, d_1, ...d_{nd-1})$.

The current vehicle's occupancy (demand) of a d_i is defined as CS_i. For this work, it is assumed that the demand of a p_i always consists of one person, but we envisage that extending this for more persons should be relatively straightforward. With this assumption, the CS_i is incremented $CS_i + 1$ when visiting a p_i and reduced $CS_i - 1$ when visiting a q_i. Each d_i has a maximum seating capacity MS of 4. If $CS_i \geq MS$, then d_i can only visit a q_i until the CS is reduced. When $CS_i < MS$, d_i can continue picking up more customers.

4.1.1 Cost Function

The objective function of the MVRRPP involves minimising the total cost, which is the sum of the distance travelled by each vehicle, the customer's waiting time and travel time.

The distance between two waypoints $dist$ is calculated using the Euclidean distance formula:

$$dist(x, y) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{4.1}$$

where (x_1, x_2) and (y_1, y_2) are two w coordinates. By establishing $dist$, the total distance travelled by a driver $dist_d$ can be calculated as follows:

$$dist_d = \sum_{i=0}^{n} dist(w_{i-1}, w_i) \tag{4.2}$$

where the distances between the previous waypoint w_{i-1} and the current waypoint w_i is calculated for all the waypoints n in d_i (until the driver returns to the u_i). We consider the distance cost of the solution as the sum of distances travelled by all drivers:

$$Cost_{dist} = \sum_{d=0}^{nd-1} dist_d \tag{4.3}$$

The customer's waiting time and travel time are represented as a function of distance. This assumption is based on the fact that the further the driver's vehicle is from the customer, the more time it will take for the customer to be picked up. The distance can be converted to actual minutes by considering the vehicle's speed. The waiting time is the time spent by the customer waiting to be served by a vehicle (the time it took for the driver to arrive at the pick-up location p_i from the origin location u_i). Since time is a function of distance, the customer's waiting time is the cumulative

distance travelled by the vehicle up to the time of pickup. The cumulative distance CD_w of a driver d_i until a certain waypoint is calculated as follows:

$$CD_w(w_z) = \sum_{i=1}^{z} dist(w_{i-1}, w_i) \tag{4.4}$$

where the distances between each w_i in d_i is calculated until the index of the specified waypoint z is reached. By formulating CD_w, the customer's waiting time WT_c can be calculated as follows:

$$WT_c = CD_w(p_i) \tag{4.5}$$

where the CD_w travelled to a customer's pickup location p_i in d_i is calculated. We then calculate the driver's total waiting time DWT_d for all customers in d as follows:

$$DWT_d = \sum_{c=0}^{nt-1} {WT_c}^2 \tag{4.6}$$

where the waiting time for each customer in d is calculated and squared until the number of customers in d, represented as nt, is reached. The waiting time is squared so that it is distributed equally between all the customers. By squaring, we are amplifying the waiting time cost of each customer; therefore, solutions that keep the waiting time low across each customer are preferred compared to solutions containing a mixture of low or high waiting times, as the high ones are amplified by the squaring, increasing the overall waiting time cost. The sum of the total waiting time cost of all drivers $Cost_{wt}$ is then calculated using the following formulation:

$$Cost_{wt} = \sum_{d=0}^{nd-1} DWT_d \tag{4.7}$$

Another objective function to consider is the travel time, the time a driver takes from a customer's pickup to the customer's drop-off (the time a customer spends inside a vehicle). The customer's travel time TT_c can be calculated as follows:

$$TT_c = CD_w(q_i) - CD_w(p_i) \tag{4.8}$$

where the customer's waiting time $CD_w(p_i)$ is subtracted from the customer's dropoff time $CD_w(q_i)$. The driver's total travel time DTT_d of all customers in d can then be calculated using:

$$DTT_d = \sum_{c=0}^{nt-1} {TT_c}^2 \tag{4.9}$$

where the travel time for each customer in d is calculated and squared until nt is reached. Similarly to the waiting time, the travel time is also squared for the same reason; to prefer solutions that distribute the travel time equally between customers. The total travel time of all drivers $Cost_{tt}$ can then be calculated as follows:

$$Cost_{tt} = \sum_{d=0}^{nd-1} DTT_d \tag{4.10}$$

The cost function's overall cost $Cost_{total}$ can therefore be calculated as follows:

$$Cost_{total} = Cost_{dist} + Cost_{wt} + Cost_{tt} \tag{4.11}$$

For evaluation purposes, we then calculate the travel impact time for each customer. This is the additional time a customer incurs due to ride-pooling, the time after pickup, where the driver may pick up additional customers before finally dropping off the customer. We first determine the optimal travel time. This is time it takes for a vehicle to travel directly from the customer's pickup location to the destination without ride-pooling. The optimal travel time for a customer OT_c is calculated as follows:

$$OT_c = dist(p_i, q_i) \tag{4.12}$$

where the distance between the customer's pickup p_i and drop-off q_i is calculated. By obtaining the OT_c, we can then determine the travel impact time IT_c of the customer using the following formula:

$$IT_c = TT_c - OT_c \tag{4.13}$$

where the optimal travel time of the customer OT_c is subtracted from the actual travel time TT_c of the same customer .

4.2 The Dataset of Problem

Peng et al. [75] generate a VRP instance by creating a set of nodes representing the depot and customers' locations. Each node contains a randomly generated 2-dimension coordinate in Euclidean space from the unit square of [0,1] x [0,1] and a randomly generated demand containing a discrete value between 1 and 9, except for the depot, which has a demand of 0.

The dataset for this project was generated similarly to that of Peng et al. [75] but adapted to include the following: multiple depots (where for the MVRRPP, we consider the depots as the origin locations), the removal of demand (since it's assumed that the demand of each pickup is always one) and the modification of the customers'

nodes (where for each customer there is a pickup and dropoff location). This dataset was generated by creating three tensors for the origin, pickup and drop-off locations using TensorFlow, an open-source library for machine learning. The origin tensor represents the drivers' starting and finishing location of a route. Each tensor has a three-dimensional shape consisting of the number of samples, the number of drivers/customers and 2D coordinates. The first dimension represents the number of samples, each representing a different MVRRPP instance. The second dimension specifies the number of drivers for the origin tensor and the number of customers for the pickup and drop-off locations since the number of pickups and drop-offs equals the number of customers. The third dimension represents a randomly generated two-dimensional coordinate (x,y) consisting of the latitude and longitude. The latitude and longitude values are randomly generated integers between 0 and 1. The three tensors are then combined to create a dataset consisting of all the coordinates needed for the MVRRPP. Different datasets were created to evaluate the MVRRPP on various scenarios containing a number of different customers and drivers. For each group of 4, 5, and 6 drivers, a different dataset was created with 20, 50, and 100 customers. Each dataset consisted of 500 samples (different MVRRPP instances). Figure 4.7 illustrates the dataset in the tensor format.

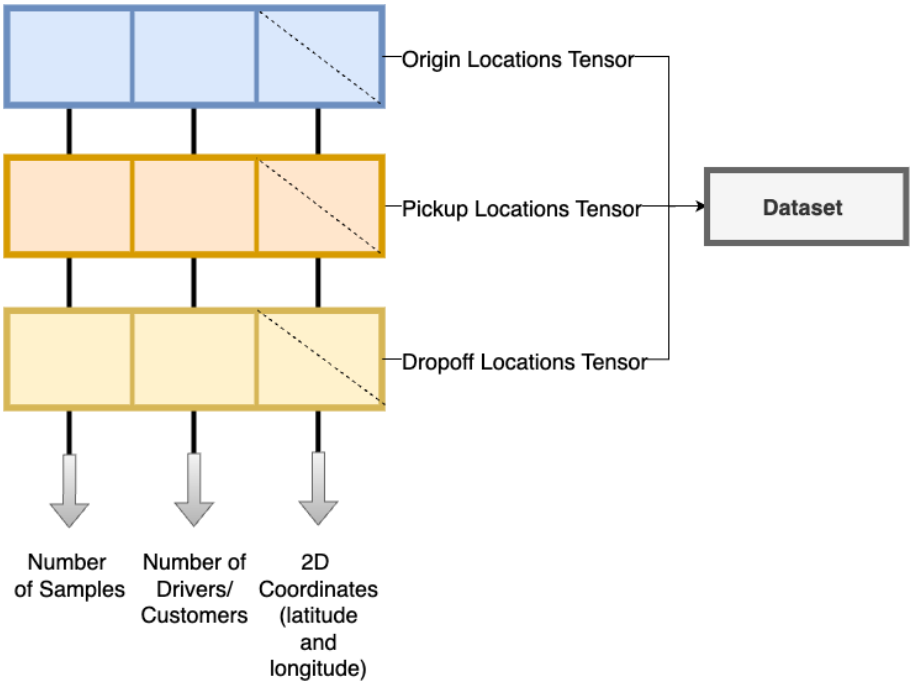


Figure 4.1 Diagram illustrating the dataset of the problem

4.3 Objective 1: Using Tabu Search as a baseline

As highlighted in Chapter 3, metaheuristic algorithms are frequently used to address different variants of the VRP. For the first objective of this research, a TS metaheuristic algorithm was implemented as a baseline to solve the MVRRPP. The performance of the implemented TS was then evaluated in terms of cost and computational times.

4.3.1 Formulating the Initial Solution

An initial solution was constructed for the TS, which involved creating the initial routes for each driver. Each route is a sequence of locations visited by a driver. To create the initial solution containing the initial routes, the customers were distributed equally to each driver's route in a round-robin manner. When assigning a customer to a driver's route, their pickup is assigned first, followed by their drop-off locations. After assigning all customers to the drivers' routes, each driver's origin location is added to the start and end of their respective routes. This indicates that each driver must start their journey from their origin location and finish at that same origin location.

Figure 4.2 illustrates the formulation of the initial solution, where for each customer in the instance, their pickup (represented by an orange circle) and dropoff (represented by a yellow circle) location were assigned to a driver. This process was repeated until all customers were distributed equally among drivers. Each driver's origin location (represented by the blue circles) was then assigned to the beginning and end of their respective routes. The diagram shows how all drivers' routes form the initial solution.

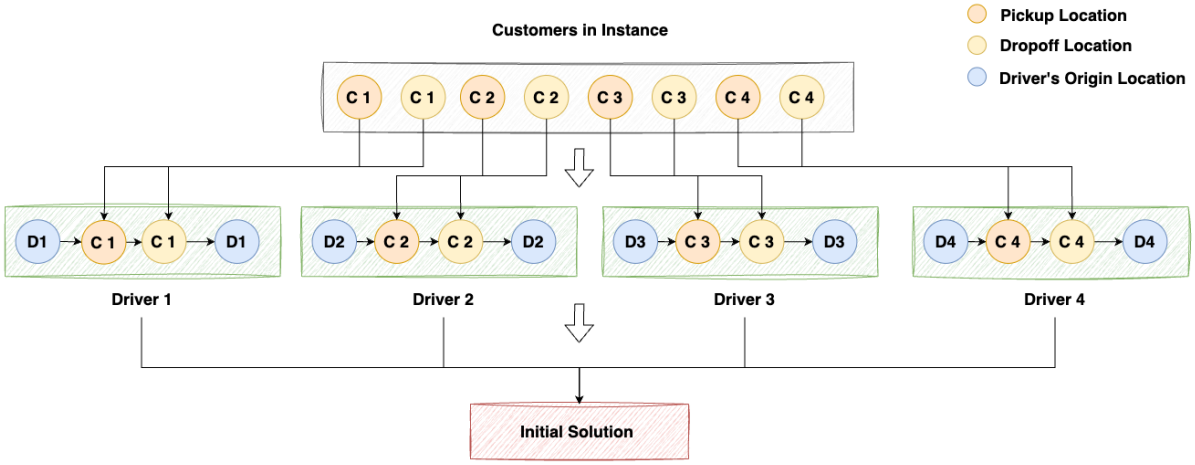


Figure 4.2 Diagram illustrating the formulation of the initial solution

4.3.2 Creating the Neighbourhood solutions

In TS, the neighbourhood solutions are a set of new solutions obtained from the current solution when modified using certain moves. The neighbourhood solutions allow the algorithm to explore the solution space and find better solutions from the current one. As previously mentioned, the solution consists of a list of driver routes, each starting and finishing at their respective origin location. Two types of moves were applied when creating the neighbourhood solutions. These moves involved swapping the pickup and drop-off nodes within the same driver's route and moving pickup and drop-off nodes to the beginning or the end of another driver's route.

The first move consisted of looping through each driver's route and swapping each route's location with every other location in that route, excluding the first and last location (origin location). After swapping a location, the new route's sequence is verified to check whether each customer's pickup location is before its respective drop-off location. If the drop-off location is before the pickup after a swap, that solution is omitted, and the algorithm proceeds with the next swap. The vehicle's occupancy is also checked after each swap to ensure its capacity has not been exceeded throughout the sequence. This involved counting the number of customers in a vehicle after a pickup. If the number of customers exceeds 4, that solution is omitted. If, after a swap, the solution satisfies each of these constraints, the newly updated route and the remaining drivers' routes are added to the list of neighbourhood solutions as one solution.

Figure 4.3 illustrates the implementation of the swap moves, where the blue circles represent the driver's origin location, the orange circles represent the customer's pickup, and the yellow circles represent the customer's dropoff. The diagram shows an example of the driver's one route and two solutions demonstrating the horizontal swap moves on this route. In the first solution, swapping the 'C1' dropoff location with the 'C2' pickup will result in a valid solution since both 'C1' and 'C2' pickups would remain before their respective dropoff locations. This solution is therefore added to the neighbourhood solutions. In the second solution, swapping the 'C1' dropoff location with the 'C2' dropoff location would result in an invalid solution since the 'C2' dropoff will now appear in the route's sequence before its corresponding pickup location.

The vertical moves involve looping through each driver's route and inserting each customer's pickup and drop-off location of that route into the beginning and end of the remaining drivers' routes. When inserting a customer's pickup and drop-off location into a new driver's route, they are removed from their original route and inserted into another driver's route. The new and remaining routes are then added as one solution to the list of neighbourhood solutions. This process is repeated until the customer's locations are inserted into each driver's route. Each insertion consists of

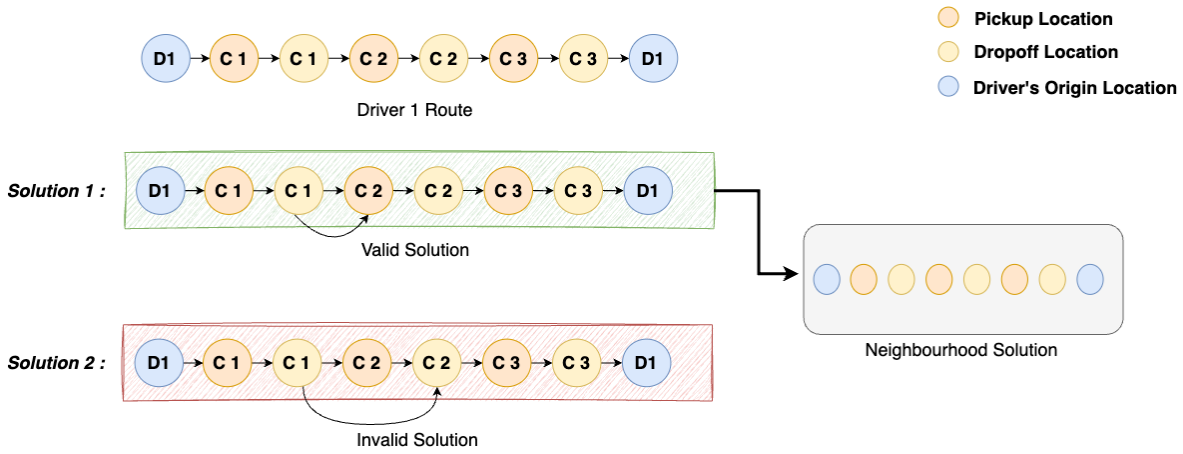


Figure 4.3 Diagram illustrating the horizontal swaps of the neighbourhood solution

adding the locations to the start (after the first location) and end of the route (before the last location), with each insertion considered as a separate solution.

The vertical moves can be formulated as follows: for the same customer c_i, the waypoint w_i consisting of the customer's pickup p_c and the waypoint w_j consisting of the customer's dropoff q_c are removed from a driver's solution d_i . The pickup-dropoff waypoints are then either:

- prepended to another driver solution d_j : d'_j = (w_i, w_j) $\oplus$ d_j
- appended to another driver solution d_j : d'_j = d_j $\oplus$ (w_i, w_j)

where $\oplus$ is the list concatenation operation, and d'_j is the new solution for driver j after the move. Since the pickup-dropoff pair is moved to the new route in the correct order (pickup before dropoff) and the driver always starts and ends the route with no occupancy, no constraints are violated after this move.

Figure 4.4 shows an example of the neighbourhood solution's vertical moves. This diagram contains the routes of two drivers: Driver 1 and Driver 2. The first customer's pickup location (orange circle) and dropoff location (yellow circle) in Driver 1's route are moved to the start of Driver 2's route after the origin location (blue circle). The new routes of Drivers 1 and 2 are then added as one solution to the neighbourhood solution.

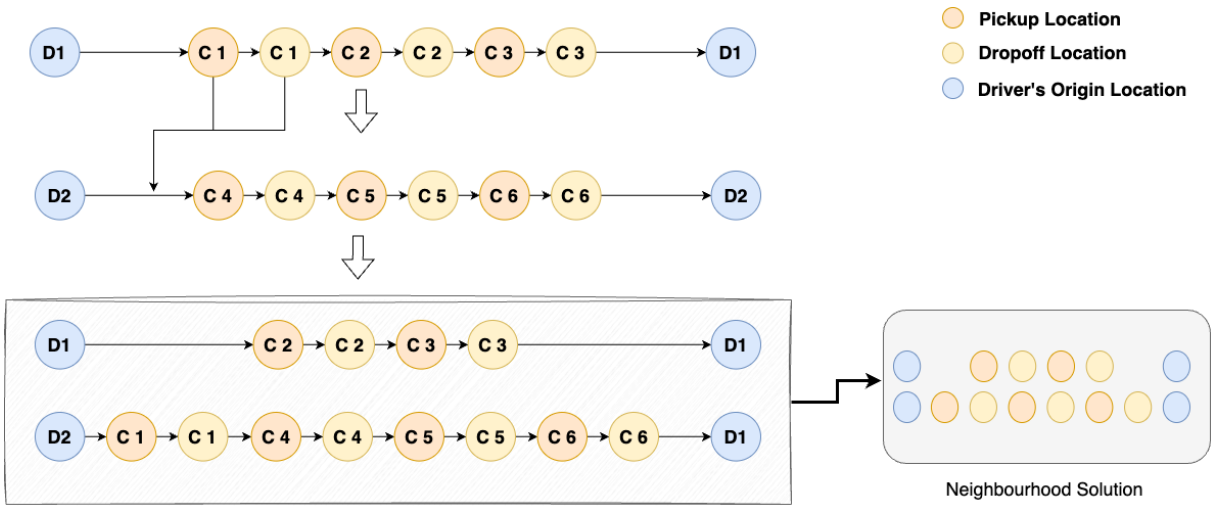


Figure 4.4 Diagram illustrating the vertical moves of the neighbourhood solution

4.3.3 Finding the Best Candidate

At the start of the algorithm, TS generates the initial solution containing each driver's route and adds it to the tabu list. Given the initial route, a set of different neighbourhood solutions are generated using the moves specified in Section 4.3.2. Each solution in the neighbourhood solution is known as a candidate and is evaluated using the cost function defined in Section 4.1.1. If a candidate has a lower cost than the current 'best candidate' and wasn't already added to the tabu list, that candidate becomes the new 'best candidate'. This new 'best candidate' is then added to the tabu list. If the tabu list exceeds its predefined size of 10 candidates, the oldest candidate will be removed. The 'best candidate' cost is then compared with the 'best solution' cost; if lower, the 'best candidate' becomes the new 'best solution'. The process is repeated for 100 iterations, where for each iteration, the current 'best candidate' of the previous iteration is used to generate new neighbourhood solutions. A stopping criteria was implemented that stops the TS from executing further iterations if the 'best solution' has not improved after 20 iterations. After executing the number of iterations, the algorithm will return the best solution found during the TS process.

The performance of TS was evaluated on multiple datasets, each including different scenarios with a different number of drivers and customers, as highlighted in Section 4.2. For each dataset, the TS algorithm was executed on 500 different instances (samples). The performance of the TS on a dataset (problem scenario) was assessed by taking the average performance of each metric on the 'best solutions' obtained across the 500 instances. This involved evaluating the average total distance, waiting time, and travel impact time. In addition, the actual time taken (in seconds) to solve each instance in the dataset was also recorded, along with the number of iterations needed to find the 'best solution'.

4.4 Objective 2: Find the appropriate representation of states, actions and reward to model as a RL Problem

For the second objective of this research, an RL algorithm using a dynamic attention model was implemented. As previously mentioned, the design of the attention model was based on research by Peng et al. [75]. We extend and adapt this work to solve the MVRRPP.

4.4.1 State

The state contains the information describing the current environment for the agent to be able to decide on the next action. The initial state encodes the coordinates of the origin, pickup and drop-off locations. Table 4.1 shows the information recorded in the state.

Table 4.1 Information recorded in the state

Observation	**Data Stored**	**Data Type**	**Dimensions**
Available Pickups	0 - Not visited , 1 - Visited	Binary	(batch size, number of customers)
Available Drop-offs	0 - Not visited , 1 - Visited	Binary	(batch size, number of customers)
Current driver's origin location	Index of origin location in action space	Int	(batch size, 1)
Number of occupied seats in the driver's vehicle	Number of customers picked up but not yet dropped off	Int	(batch size, 1)
Driver's Distance Travelled	Total distance travelled in a route	Float	(batch size, 1)
Customer's Waiting Time	The waiting time of each customer in a route	Float	(batch size, number of customers)
Customer's Travel Time	The travel time of each customer in a route	Float	(batch size, number of customers)

The state records the available customers' pickup and drop-off locations. Visited locations are updated using binary values where a value of '1' indicates that the location has been visited, while a value of '0' indicates that it has not been visited yet. Keeping a record of the visited locations allows us to create a mask (explained further in the decoder subsection) that prevents the agent from revisiting locations. The state also keeps track of the current driver's origin location, which is also used for masking

purposes to prevent the agent from selecting a different origin location when returning to the starting location. The state stores the number of occupied seats of the current driver's vehicle to prevent the agent from selecting further customers while the vehicle is full. The state also stores each driver's travelled distance, customer's waiting time, and travel time to calculate the reward function (explained further in Section 4.4.3), which is used to determine the performance and quality of the solution.

Following a similar approach to Peng et al. [75], a dynamic encoder-decoder architecture was implemented where, during different steps, the feature embeddings of the nodes are dynamically updated based on the current instance. The following subsections include a detailed explanation of the dynamic encoder-decoder architecture of the model. Figure 4.5 illustrates the data flow in the encoder-decoder architecture, starting from the graph instance.

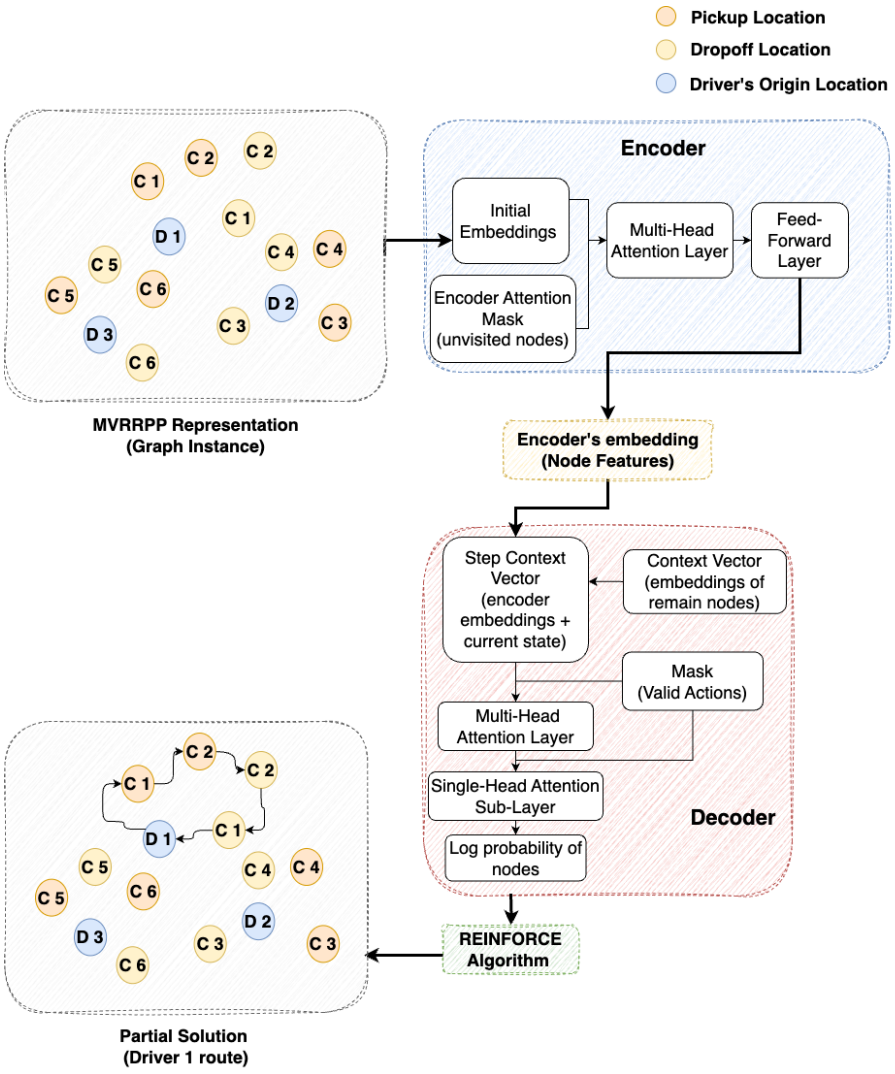


Figure 4.5 Diagram illustrating the data flow of the encoder-decoder architecture

State Encoder

An encoder was implemented using a GAT, whose purpose is to encode the information of the MVRRPP representation to an embedding. The MVRRPP representation consists of the nodes' coordinates, organised as a tuple of 3 tensors, each containing the waypoint w_i coordinates for the origin points U, pickups P and drop-off locations Q. Each waypoint node w_i has dimension of $d_x = 2$. For each tensor, the encoder uses a dense layer in the GAT that takes the input data of the tensor and forms the initial embeddings for that tensor.

By carrying out hyperparameter optimisation before training (explained in more detail in Section 4.4.5), the optimal set of hyperparameters that minimised the cost function was found. One of these hyperparameters included the optimal embedding dimension size d_h. The dense layers use this optimal embedding dimension size d_h to specify the number of output units for the initial embedding (the output space dimensionality). The dense layers apply linear transformations using learnable parameters for the weights $W \in \mathbb{R}^{d_h \times d_x}$ and bias $b \in \mathbb{R}^{d_h}$. Each tensor has it's own parameters: W_u and b_u for the origin coordinates, W_p and b_p for the pickup coordinates and W_q and b_q for the drop-off coordinates. During training, the W and b of these layers are adjusted to capture patterns in the input data that are beneficial for solving the MVRRPP. The initial node embedding $h_i^{(0)}$ can be represented using the following function:

$$h_i^{(0)} = \begin{cases} W_U x_i + b_U & \text{if } i \in U \\ W_P x_i + b_P & \text{if } i \in P \\ W_Q x_i + b_Q & \text{if } i \in Q \end{cases} \tag{4.14}$$

Each embedding is concatenated to form the initial embeddings of the graph. The initial embeddings and an attention mask are then passed through three stacked attention layers $S = 3$, each containing a multi-head attention (MHA) sublayer and a fully connected feed-forward (FF) sublayer. The attention layers aim to capture the relationships between nodes using an attention mechanism that assigns the attention weights to different parts of the input sequence. In the attention layer $\ell \in \{1, ..., S\}$, each layer's node embedding is represented as h_i^ℓ, where each node is i. The output of layer $\ell - 1$ is $\left\{h_0^{(\ell-1)}, ..., h_s^{(\ell-1)}\right\}$ which also represents the input of ℓ.

The attention mask guides the multi-head attention mechanism to focus on unvisited nodes and exclude nodes that were already visited in the current partial solution $\pi_{1:t-1}$, where $t \in \{1, ..., T\}$ represents each step. It dictates which nodes the attention mechanism can attend to in the GAT. An attention mask (represented as a binary matrix) was created, where the 0s indicate locations that should be considered (yet to be visited), and 1s indicate locations that should be ignored (already visited).

The attention mask is updated whenever a driver returns to their origin location to reflect the current state of the visited origin, pickups, and drop-off locations.

Each multi-head attention sub-layer uses 8 attention heads $M = 8$, where each attention head is represented as $m \in \{1, ...M\}$. The initial embedding tensor is used for the queries $r_{im}^{(\ell)} \in \mathbb{R}^{d_k}$, keys $k_{im}^{(\ell)} \in \mathbb{R}^{d_k}$, and values $v_{im}^{(\ell)} \in \mathbb{R}^{d_v}$ tensor. They are projected into several subspaces using linear transformations by multipling each tensor with their respective weight matrices $W_m^R \in \mathbb{R}^{d_k \times d_h}$, $W_m^K \in \mathbb{R}^{d_k \times d_h}$ and $W_m^V \in \mathbb{R}^{d_v \times d_h}$, where d_k = d_v = $\frac{d_h}{M}$, with d_k representing the dimensions of key and d_v representing the dimensions of value. These transformed tensors are then split into multiple heads to execute the attention computations simultaneously across each head in parallel. The r, k, and v vectors for each attention head m in the multi-head attention layer ℓ, can be represented as follows:

$$r_{im}^{(\ell)} = W_m^R h_i^{(\ell-1)}, k_{im}^{(\ell)} = W_m^K h_i^{(\ell-1)}, v_{im}^{(\ell)} = W_m^V h_i^{(\ell-1)} \tag{4.15}$$

The compatibility scores between the r and k matrices are computed using the matrix multiplication operation. The compatibility scores capture the similarity between the r and k, which are required for determining the attention weights. These compatibility scores are then rescaled by dividing the scores with the square root of the dimension of k ($\sqrt{d_k}$) to control the magnitude of the compatibility matrix. The attention mask specified earlier is then applied to the compatibility scores, by assigning a negative infinity value to visited nodes and therefore preventing attention from being assigned to masked positions (visited nodes). The compatibility scores can be formulated as follows:

$$u_{ijm}^{\ell} = \begin{cases} \left(r_{im}^{(\ell)}\right)^T k_{jm}^{(\ell)} & \text{if } x_j \notin \pi_{1:t-1} \text{ or } j \in U \\ -\infty & \text{otherwise} \end{cases} \tag{4.16}$$

where if node x_j is not in the partial solution $\pi_{1:t-1}$ or node j is an origin node in the set of origin points U, the compatibility scores are calculated, otherwise an negative infinity value is assigned.

The softmax function is applied to the compatibility scores to obtain the normalized attention weights for each element of the input sequence. This can be formulated using the following formula:

$$a_{ijm}^{(\ell)} = \frac{e^{u_{ijm}^{(\ell)}}}{\sum_{j'=0}^{n} e^{u_{ij'm}^{(\ell)}}} \tag{4.17}$$

where the softmax function is calculated by finding the exponential of the attention score $e^{u_{ijm}^{(\ell)}}$ and dividing it over the sum of the exponential of the attention

scores of all nodes j', defined as $\sum_{j'=0}^{n} e^{u_{ij'm}^{(\ell)}}$.

The attention weights are multiplied by $v_{im}^{(\ell)}$ to calculate the weighted sum of v for each position in the r sequence. These attention weight allow the attention mechanism to attend to specific parts of the input data and capture the relevant information from the input sequence. The weighted sum of v can be represented as follows:

$$h_{im}^{'(\ell)} = \sum_{j=0}^{n} a_{ijm}^{(\ell)} v_{jm}^{(\ell)} \tag{4.18}$$

For each m, the weighted sum of v, $h_{im}^{'(\ell)}$ is passed through a dense layer, where linear transformation is applied using the weight of the respective m head, $W_m^Y \in \mathbb{R}^{d_h \times d_v}$. The sum of all m heads outputs represents the output of the multi-head attention layer. The output of the multi-head attention layer for node i at layer ℓ can therefore be represented as:

$$\mathsf{MHA}_i^{(\ell)}\left(h_0^{(\ell-1)}, ..., h_n^{(\ell-1)}\right) = \sum_{m=1}^{M} W_m^Y h_{im}^{'(\ell)} \tag{4.19}$$

The initial embeddings and the multi-head attention layer output are combined using a skip connection. The purpose of the skip connection is to mitigate the vanishing gradient problem during training as gradients backpropagate through multiple layers. The skip connection allows the gradients to flow directly to subsequent layers, helping it to preserve information from the original input (the input embeddings).

The output of the skip connection is then passed through a hyperbolic tangent activation function (tanh), where non-linearity and normalization (within the range of -1 to 1) are applied. This can be represented using the following formula:

$$\hat{h}_i^{(\ell)} = \mathsf{tanh}\left(h_i^{(\ell-1)} + \mathsf{MHA}_i^{(\ell)}\left(h_0^{(\ell-1)}, ..., h_n^{(\ell-1)}\right)\right) \tag{4.20}$$

After the tanh activation, the output is processed through two feed-forward layers. Each feed-forward layer applies a linear transformation followed by an activation function, each using different trainable parameters: $W_0^F \in \mathbb{R}^{d_F \times d_h}$ and $b_0^F \in \mathbb{R}^{d_F}$ for the first feed-forward layer and $W_0^F \in \mathbb{R}^{d_F \times d_h}$ and $b_1^F \in \mathbb{R}^{d_F}$ for the second feed-forward layer, where $d_F = 4 \times d_h$. The rectified linear unit (ReLU) activation is applied after the first feed-forward layer, which applies non-linearity, removing any negative values and replacing them with zeros. This is outlined through the formula presented below:

$$\mathsf{FF}\left(\hat{h}_i^{(\ell)}\right) = W_1^F \mathsf{ReLu}\left(W_0^F \hat{h}_i^{(\ell)} + b_0^F\right) + b_1^F \tag{4.21}$$

The tanh activation is then applied after the second feed-forward layer, which is the final output of the attention layer. The output of a node after passing through the multi-head attention layer and the feed-forward sublayer can be represented as follows:

$$h_i^{(\ell)} = \text{tanh}\left(\hat{h}_i^{(\ell)} + \text{FF}\left(\hat{h}_i^{(\ell)}\right)\right) \tag{4.22}$$

The output of **S** stacked attention layers represents the output of the encoder. The final node embedding h_i^S for node i after S attention layers is formulated as follows:

$$h_i^S = \text{ENCODE}_i^S\left(h_0^0, ..., h_s^0\right) \tag{4.23}$$

Decoder

A decoder was implemented to generate the solution sequence for the MVRRPP. The decoder is responsible for selecting a node at each step. Similarly to the encoder, the decoder used multi-head attention to allow the model to learn complex relationships and capture higher-level features in the context of the given decoding step. A step context vector was created using the embeddings generated by the encoder and the current environment's state. This vector captures information about the MVRRPP's current state, which is essential for making decisions in the subsequent steps of the decoder stage.

The step context vector consists of the previous node's embeddings $h^S_{\pi_{t-1}}$ if at step $t-1$ or the first chosen origin point embedding h_O^S if at step $t=1$, the remaining seating capacity of the current vehicle RS_t, the total distance travelled of all vehicles TD_t, the number of drivers left TO_t, the number of unpicked customers TC_t, the indexes of the remaining drivers RD_t and the indexes of the remaining customers' pickup RP_t and dropoff RQ_t. The step context vector $h_c^{'}$ is formulated using the following formula:

$$h_c^{'} = \begin{cases} \left[\bar{h}_t; h_O^S; RS_t; TD_t; TO_t; TC_t; RD_t; RP_t; RQ_t;\right] & \text{if } t = 1 \\ \left[\bar{h}_t; h^S_{\pi_{t-1}}; RS_t; TD_t; TO_t; TC_t; RD_t; RP_t; RQ_t;\right] & \text{if } t > 1 \end{cases} \tag{4.24}$$

where $\bar{h}_t$ represents the mean graph embeddings of unvisited nodes and $;$ represents the concatenation in the step context. The step context vector is passed through a dense layer, which applies a linear transformation using a learned weight matrix at each decoding step. The output of this layer projects the step context vector into a new space compatible with the decoder's attention mechanism. This projected step context vector is then added to the context vector $Q\_context$.

The $Q\_context$ consists of the mean embedding for the remaining nodes. It is

achieved by dividing the encoder's embeddings with the number of remaining available nodes (origin, pickup, and dropoffs) and passing it through a dense layer for linear transformation. Adding the context vector with the step context vector results in the final query tensor r for the decoder's multi-head attention. This r vector provides context information on the remaining unvisited nodes and the agent's current state. The r vector was split into M attention heads to allow the parallel processing of the multi-head attention computations, where each head attends to different aspects of the query vector.

The decoder attention mechanism's key k and value v projections were calculated by passing the encoder's embedding h_j^S through dense layers allocated for k and v, and applying a linear transformation with the weight matrix W^K and W^V respectively. These parameters are different from the ones used in the encoder. The k and v tensors are each split into M heads for parallel processing. The decoder's r, k and v can be formulated using the following formula:

$$r\left(c\right)_m = W_m^R h_c^{'},\ k_{jm} = W_m^K h_j^S,\ v_{jm} = W_m^V h_j^S \tag{4.25}$$

where $r\left(c\right)_m$ represents the query vector for the context vector $h_c^{'}$ in head m and k_{jm} and v_{jm} represent the key and value respectively for the current node j in head m.

A mask was created to guide the decoder's attention mechanism. The mask ensures the agent selects a valid node as its next action by excluding (masking out) invalid actions that do not adhere to the rules and constraints of the MVRRPP. Upon selecting the initial action, the mask excludes all nodes except the origin locations, allowing the agent to select an origin point to start the route. Once the agent selects an origin location, all origin locations are masked out, and the available pickup locations are unmasked. The remaining origin locations are masked until the agent returns to the selected origin location. Once the agent selects a customer's pickup location, that pickup location is masked, and the respective customer's drop-off location is unmasked. At this stage, the mask allows the agent to select another pickup location or drop off the picked-up customer.

When an agent visits a pickup location, the occupancy is incremented by 1. If the vehicle's occupancy reaches capacity (which was set to 4), all available pickup locations are masked out to prevent the agent from picking up further customers. If a customer is dropped off, the occupancy is decreased by one, and the available pickup locations are again unmasked since the vehicle can now pick up more customers. If, at a particular step, there are no drop-off locations left because the customers picked up have been all dropped off or due to all customers in the instance having been dropped off, then the selected origin location becomes unmasked, allowing the agent to return to the origin location.

When an agent returns to the selected origin location, that origin location becomes masked along with the pickup locations (if there are any left). The remaining origin locations are then unmasked, allowing the agent to select another origin location for the next route. This process is repeated until all customers have been dropped off or until there are no further origin locations from which to start a new route. The mask is of a boolean type, where valid nodes have 'False' values whilst invalid nodes have 'True' values. The decoder's mask DM can be represented as a list of boolean values dm_i, where the size of the list is equivalent to the total number of nodes in the instance, tn (number of drivers nd + number of pickup and dropoffs $2(nc)$). The DM can be formulated as follows:

$$DM = (dm_1, dm_2, ...dm_{tn}) \tag{4.26}$$

The decoder's multi-head attention is computed using its r, k, and v vectors and mask DM. Similarly to the encoder, the compatibility scores are calculated by finding the matrix multiplication between r and v, followed by dividing the square root of the dimension of the head depth. The decoder's mask is then applied to the compatibility scores, where the elements equivalent to a 'True' value (indicating a masked node) are assigned a negative infinity value. The compatibility score between the transposed query vector $r^T_{(c)m}$ and the key k_{jm} of node j in head m can be formulated as follows:

$$r_{(c)jm} = \begin{cases} r^T_{(c)m} k_{jm} & \text{if } dm_j = False \\ -\infty & \text{otherwise} \end{cases} \tag{4.27}$$

where the compatibility score is calculated if dm_j, representing the index of node j in the decoder's mask DM is 'False' otherwise a negative infinity value is assigned.

The attention weight of each element $a_{(c)jm}$ in the input sequence was then calculated by applying a softmax function to the masked compatibility scores. By applying the softmax function, the negative infinity values in the masked compatibility score resulted in a zero probability (weight) and, therefore, will not influence the attention distribution. The attention weight for node j can be formulated using the following formula:

$$a_{(c)jm} = \frac{e^{u_{(c)jm}}}{\sum_{j'=0}^{tn} e^{u_{(c)j'm}}} \tag{4.28}$$

The weighted sum of the v vectors for each element in query r was then determined by applying a matrix multiplication between the attention weights $a_{(c)jm}$ and the v_{jm} vectors. These attention scores are then reshaped to include information from all attention heads. The weighted sum $h'_{(c)m}$ can be formulated as follows:

$$h'_{(c)m} = \sum_{j=0}^{tn} a_{(c)jm}\, v_{jm} \tag{4.29}$$

The final output of the decoder's multi-head attention is calculated by applying a linear transformation using its learnable weight W_m^Y . This output contains the context information for the current decoding step. The output of the decoder can be presented using the below formula:

$$h_c = \sum_{m=1}^{M} W_m^Y\, h'_{(c)m} \tag{4.30}$$

The next stage of the decoder is to calculate the log probability when selecting each node in the MVRRPP. The log probability is determined using a single-head attention sub-layer with the following inputs: the multi-head attention mechanism's output (represented as $r = W^R h_c$), the projection of node embeddings (represented as $k_j = W^K h_j^S$) and the decoder's mask DM. The compatibility scores were first calculated using the dot product between the r and k vectors, followed by dividing the square root of the output embedding size. The tanh activation function was then applied element-wise to the compatibility scores, where it was then scaled by the tanh clipping size $cl = 10$. The decoder's mask is then applied to these compatibility scores to prevent the model from assigning probabilities to invalid nodes. The compatibility score for node j at step t can be formulated as follows:

$$u_j = \begin{cases} cl.\tanh\left(r^T k_j\right) & \text{if } dm_j = False \\ -\infty & \text{otherwise} \end{cases} \tag{4.31}$$

where if the index of node j in the decoder's mask dm_j is 'False', the compatibility score is calculated otherwise if dm_j is 'True', a negative infinity value is assigned. After the compatibility scores are calculated, a softmax is applied to obtain the final log probabilities for each node. Given the current input instance X and the partial solution until step $t-1$ (represented as $\pi_{1:t-1}$), the probability of selecting a specific node x_j to visit at step t can be determined using the following formula:

$$p_\theta\left(\pi_t = x_j \,|X, \pi_{1:t-1}\right) = \frac{e^{u_j}}{\sum_{j'=0}^{tn} e^{u_{j'}}} \tag{4.32}$$

Depending on the decoding type (further explained during Section 4.4.6), the log probabilities are used to select the node for the current step. The decoding type can either be 'greedy' or 'sampling'. If the decoding type is 'greedy, ' the node with the highest probability is selected, while if the decoding type is 'sampling', a node is

randomly chosen from the probability distribution. This selected node is then added to the current partial solution.

Dynamic Encoder-Decoder

The implemented encoder and decoder dynamically updates the feature embeddings of the nodes at different steps depending on the current instance. When a node is selected from the predicted distributions generated by the decoder, that node is added to a partial solution. The partial solution is equivalent to a vehicle's route, which consists of the nodes visited by that vehicle, starting at the origin location and ending when it returns to its origin location. The remaining nodes not included in this partial solution are treated as a new instance. Solving this new instance is equivalent to constructing a new route from a different origin location. When a vehicle leaves from a different origin location, the encoder's attention mask is updated to reflect the visited nodes, focusing its attention on the unvisited nodes. The embeddings and context vectors are also updated based on the new attention mask that reflects this new state. Figure 4.6 illustrates how, after generating a partial solution (completing a driver route), a new instance is created (without the nodes of the partial solution) and used to update the feature embeddings through the encoder-decoder dynamically.

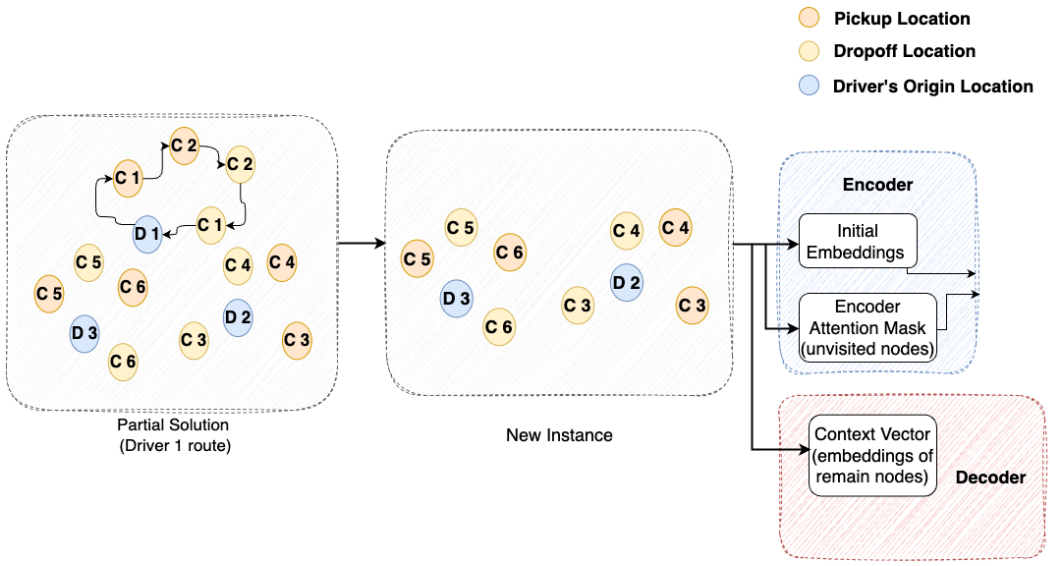


Figure 4.6 Diagram illustrating the data flow when the encoder-decoder dynamically updates the feature embeddings

The encoder's embedding h_i^t of node i at step t can be updated using the following formula:

$$h_i^t = \begin{cases} \text{ENCODER}_i^S\,(h_0^0, ..., h_s^0) & \text{if } \pi_{t-1} = u \in U \\ h_i^{t-1} & \text{otherwise} \end{cases} \tag{4.33}$$

where if node i is an origin point u, the embeddings are updated, otherwise the encoder's embeddings remains the same. The projection matrices used in the decoder's attention, such as the step context and decoder's K values, are also updated to reflect the changes in the embeddings and context vectors.

4.4.2 Actions

An action space is a set of possible actions an agent can take in a given environment. In this model, the action space consists of the possible locations that an agent can choose to visit next. There are three types of nodes in the action space: the origin, pickup, and drop-off nodes. The selected node represents the next action in the sequence of a vehicle, determining the vehicle's route. As mentioned during the decoder stage, a mask was applied to the action space to define which actions are valid for the agent to select from. The mask changes at each step, given the current state and problem constraints. The action space is represented as a discrete set of node indices. The first nu indices of the action space represent the origin locations u, the subsequent nc indices represent the pickup locations p, and the remaining nc indices represent the dropoff locations q. Given the p_i index, it's corresponding q_i index may be determined by using the following formula: $q_i = p_i + nc$.

Figure 4.7 illustrates the action space with the origin, pickup and dropoff locations.

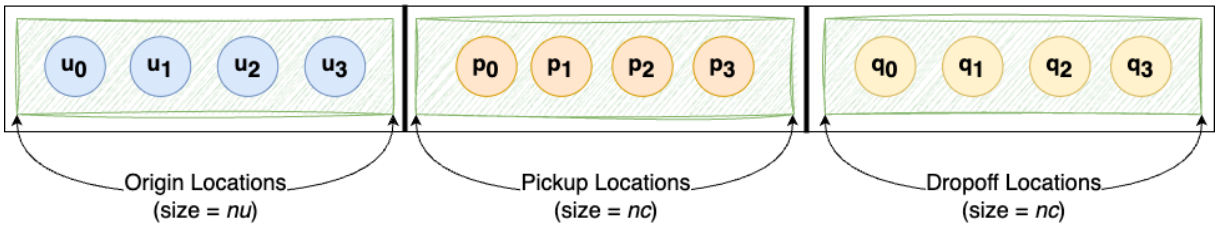


Figure 4.7 Diagram illustrating the action space

4.4.3 Reward

The reward function consisted of calculating the cost function $Cost_{total}$ (defined in Equation 4.11) and adding a penalty if any customers were left unpicked at the end of an instance. In the initial tests, the agent discovered that by picking up fewer customers, overall costs could be reduced since the agent wouldn't incur distance, waiting time, and travel time costs. This approach, however, resulted in incomplete solutions as many customers were left unpicked. To address this issue, a significant penalty was added to the reward function to ensure that the penalty for leaving customers unpicked outweighed any potential savings from not picking up customers.

The penalty for the number of customers left unpicked consists of multiplying the number of unpicked customers φ by 10000, a value that aims to magnify the penalty for leaving customers unpicked (unassigned to a driver route). The reward function can therefore be defined as:

$$R = Cost_{total} + 10000\varphi \tag{4.34}$$

4.4.4 Episodes

For each episode during training, a random dataset containing 500 instances is generated and split into batches to allow multiple instances to run concurrently. Training the model on batches increases memory efficiency and faster convergence, as the model's parameters are updated after every batch of samples, allowing it to adjust the parameters in the direction of the gradient more frequently (explained further in Section 4.4.6).

Each instance in a batch consists of randomly positioned pickup, dropoff and origin locations. In the initial state of each instance, the drivers do not have a route assigned to them, and the customers are yet to be allocated a driver. Unlike the TS experiment, in which the customers already had a driver assigned to them in the initial solution, there are no routes in the RL's initial state.

In the first step, the agent must select one of the origin locations to start a route. At each step, the agent selects a location, which can be a pickup, dropoff, or origin location, depending on the masking (explained in Section 4.4.1), which guides the agent in selecting valid actions for the current state. An agent completes a route when he returns to the chosen origin location, from which the agent must then select one of the remaining origin locations to start its next route.

An instance ends if one of the following conditions occurs: there are no more remaining customers in the instance, or there are no more drivers (origin locations) to start a new route to pick up the remaining customers. The instance's final solution represents each driver's route, which consists of the sequences of actions (locations) the agent took at each step.

After completing all instances in the current batch, the reward function (defined in Equation 4.34) then calculates the cost for each instance's final solution. For each instance, the reward function calculates the cost of each driver's route and the cost of the remaining customers left unpicked in an instance. The final cost is then used to calculate the REINFORCE loss, which involves finding the difference between the cost of the current batch and the estimated cost of the baseline model (explained further in Section 4.4.6). Depending on the gradient of the REINFORCE loss, the model's parameters are then updated. This process is repeated for all batches in the dataset. An

episode ends once all batches in the dataset are completed.

4.4.5 Hyperparameter Optimisation

Before training the REINFORCE model, Optuna, a hyperparameter optimisation framework, was used to find the optimal set of hyperparameters that minimises the reward function specified in Equation 4.11. The hyperparameters to be optimised were defined in the search space and consisted of the embedding dimension size, learning rate, and batch size.

The embedding dimension size used for the encoder and decoder was defined in the search space using the following values: 32, 64, 128, 256. The learning rate, representing the magnitude by which the parameter values are adjusted during training, was defined in the search space using the values: 0.0001, 0.001, 0.01, 0.1. When the dataset consists of many nodes (origin + pickup + drop-off locations), memory exhaustion occurs during the execution of the multi-head attention layer. Due to the GPU's memory limitations and constraints in this layer, different batch size values had to be assigned to the search space based on the number of customers in the given dataset. These values were assigned to match the tensor sizes and memory limits. Due to each dataset containing 500 samples, the batch size values defined in the search space also had to be factors of 500. The following batch size values were defined in the search space for datasets with 20 customers: 50, 100, 125. For datasets with 50 and 100 customers, the following values were defined in the search space: 20, 25, 50.

Optuna uses an objective function to determine which set of hyperparameters, defined in the search space, minimises the reward function. The objective function consists of the logic required to train the RL model using a set of suggested hyperparameters provided by Optuna during each trial. During the hyperparameter optimisation, Optuna iteratively runs trials with different sets of hyperparameters to find which hyperparameters minimise the reward function. It evaluates each set of hyperparameters using the cost returned by the objective function, which consists of the average reward of all training epochs on the given set of hyperparameters. Optuna uses the cost to evaluate the model's performance on this set of hyperparameters. Due to each trial being time-intensive because of the model's complexity, the number of trials used for the hyperparameter optimisation was set to 15. During each trial, the model was trained for 15 epochs.

The hyperparameter optimisation process was carried out for each scenario, consisting of a different number of customers and drivers. Table 4.2 shows the optimal hyperparameters achieved using Optuna for 20 customers across 4, 5, and 6 drivers, while Table 4.3 and Table 4.4 show the optimal hyperparameters achieved for 50 and 100 customers, respectively.

Table 4.2 Optimal Hyperparameters for the RL models with instances of 20 Customers

20 Customers	Drivers		
	4	5	6
Embedding Dimension Size	256	64	64
Learning Rate	0.0001	0.0001	0.0001
Batch Size	50	100	100

Table 4.3 Optimal Hyperparameters for the RL models with instances of 50 Customers

50 Customers	Drivers		
	4	5	6
Embedding Dimension Size	32	128	128
Learning Rate	0.0001	0.0001	0.0001
Batch Size	20	50	50

Table 4.4 Optimal Hyperparameters for the RL models with instances of 100 Customers

100 Customers	Drivers		
	4	5	6
Embedding Dimension Size	128	32	64
Learning Rate	0.0001	0.0001	0.0001
Batch Size	25	20	20

4.4.6 Training the REINFORCE Algorithm

The REINFORCE algorithm is a policy gradient algorithm that aims to lower costs (higher reward) by adjusting the policy's parameters. For this model, the REINFORCE algorithm was trained with a baseline model. The purpose of this baseline model was to estimate the expected cost associated with taking a sequence of actions under the model's current policy. The baseline serves as a reference point for comparison against the actual cost. The difference between the actual cost achieved from the trained model and the baseline value determines the REINFORCE loss. The gradient of this loss is used to adjust the model's parameters, increasing the likelihood of actions that lead to higher rewards. These likelihoods represent the probabilities that the agent will

take each possible action in a particular state. This baseline model can help reduce the variance in the policy gradient estimates.

The baseline model was initialised by creating a model based on the initial training model's parameters. A new dataset was then generated for the baseline model. The baseline model was evaluated on this dataset using a greedy decoder type to calculate the average cost of the current baseline model. During each epoch, a new random dataset was created using 500 samples, each including nu number of origin nodes, nc number of pickups and nc number of drop-off nodes.

During the initial training epochs, the baseline model goes through a warm-up phase, where the model is gradually introduced to the dataset to avoid potential issues like instability or divergence that might arise from immediate exposure to the entire dataset. The alpha parameter controls the warm-up phase. If the parameter is below 1, the model is in the warm-up phase, and therefore, the model skips the computation of baseline values. In contrast, if the alpha parameter is above 1, the warm-up period is completed, and the baseline values are calculated for each sample in this dataset using the decoder set to 'greedy' for a greedy execution of the baseline model. These baseline values are used further on to compare performance and reduce the variance of the REINFORCE's policy gradient estimator.

Each epoch's dataset is split into batches using an optimal batch size specified during the parameter optimisation, as explained previously in Section 4.4.5. The REINFORCE algorithm evaluates each batch separately, using each batch's instances as input data. The model evaluates each batch by iterating through the decoding steps until the state of each instance in the batch is marked as finished (based on the conditions specified in Section 4.4.4). For each instance in the batch, the model then calculates the total costs of the final solution and the log-likelihood of the selected actions.

To calculate the REINFORCE loss for the current batch, we first determine the baseline values using the current batch as input and the total costs achieved from evaluating the model on the current batch. If no baseline values are precomputed, the baseline values are calculated based on three conditions. If the alpha parameter is 0 (warm-up has not started), the baseline values are the exponential moving average (EMA) cost of the previous batches; if there are none, it is the mean of the current batch's cost. If the alpha parameter is less than 1 (warm-up phase), the combination of the EMA and baseline cost is calculated. If the alpha parameter equals 1 (warm-up phase completed), the baseline value is the combination of the EMA set to 0 and the baseline cost. The baseline values are excluded from the gradient computation to prevent any gradient from flowing through them during backpropagation.

The REINFORCE loss is then calculated by finding the difference between the total cost (achieved from evaluating the model) and the estimated baseline value. The

difference is then multiplied by the log-likelihood obtained earlier to weigh the probability of each action being chosen by the model's policy. The REINFORCE loss represents how well the current policy performed on a given set of inputs relative to the baseline estimate. The gradient of the REINFORCE loss was calculated with respect to the model's trainable parameters, which consisted of the weights of the dense layers used earlier for the encoder and decoder. The gradient of the REINFORCE loss determines the direction and magnitude of adjustments needed for each parameter in the model to improve its performance relative to the baseline. Once the gradients of the REINFORCE loss are calculated, they are then passed to the Adam optimizer, which is an extension of the gradient descent method. The Adam optimizer applies the gradients to their corresponding model's parameters to improve the policy performance.

After training the model on all batches, the model is evaluated on the baseline dataset to determine the mean cost of all instances. The difference between the mean cost of the training model and the mean cost of the baseline model is then calculated to determine if the training model performed better or worse than the baseline model. If the difference is negative, this indicates that the training model performed better. A statistical test to confirm the significance of improvement in the trained model is carried out, and if the improvement is significant, the baseline model is updated. The baseline model is updated by copying the weights from the trained model to create a new baseline model. A new dataset for the new baseline model is also created to prevent possible overfitting. The new baseline model is evaluated on this dataset using a greedy decoder to determine a new mean baseline cost.

The above training procedure is repeated for 100 epochs, allowing the REINFORCE algorithm combined with the baseline model to optimise its parameters and find a policy that minimises overall costs. The policy represents the model's ability to generate optimal solutions for the MVRRPP based on the given data. For each scenario containing a different number of customers and drivers, a separate model was used for training using the respective dataset of that scenario, as defined in Section 4.2.

4.4.7 Comparing REINFORCE with Tabu Search

The trained REINFORCE models were evaluated on their respective validation sets. These validation sets are the same datasets used to evaluate the TS algorithm on scenarios with different number of customers and drivers. Each instance (sample) in the validation set represents a different MVRRPP instance on which the trained model is tested on. A final solution was generated when testing the trained RL model on each instance in the validation set. The final solution consisted of the sequences of actions taken in the environment at each step for that corresponding instance. It represents all

the driver routes and the order in which the customers' pickup and drop-off locations were visited. The performance of the trained model was evaluated using the final solutions generated for each instance in the validation set. The metrics for evaluating each instance's final solution consisted of the waiting time, travel impact time, driver's distances, and time taken to solve the instance. For each scenario of customers and drivers specified in Section 4.2, the metrics achieved using RL were compared to the results achieved using TS on the same dataset.

When evaluating RL and TS experiments in different scenarios, the RL experiment used the pre-trained model to find solutions for unseen scenarios without needing to retrain. In contrast, TS must start the search process for each unseen scenario from the beginning.

4.5 Objective 3: Evaluate the effect of combining a RL approach with optimisation based on Tabu Search

The third objective of this research consisted of using the output achieved in the second objective as the initial solution for the TS. This objective aims to determine whether combining RL with TS can enhance the optimisation process and achieve better results in terms of solution generation time and quality.

While the initial solution in Experiment 1 was generated by distributing the customers equally across each driver's route, in this experiment, the RL solution is used as the initial solution for the TS to generate the initial neighbourhood solutions. Figure 4.8 illustrates the data flow of the TS with RL model.

As previously mentioned in Section 4.4.7, the RL solution obtained from the previous experiment includes the sequence of visited locations for each driver's route, achieved after testing the trained RL model on the validation set.

For this experiment, the TS algorithm used the same tabu list size and was executed on the same number of iterations as Experiment 1. To evaluate the impact of using the RL solution as the initial solution, TS was evaluated on the same dataset used to evaluate the algorithms in the previous experiments. By using the same dataset, iterations, and tabu list size, we were able to compare the metrics for all the experiments under the same conditions.

This process was repeated for each scenario of customers and drivers (specified in Section 4.2), where their respective RL solution was used as the initial solution and evaluated using TS separately on the dataset of that scenario.

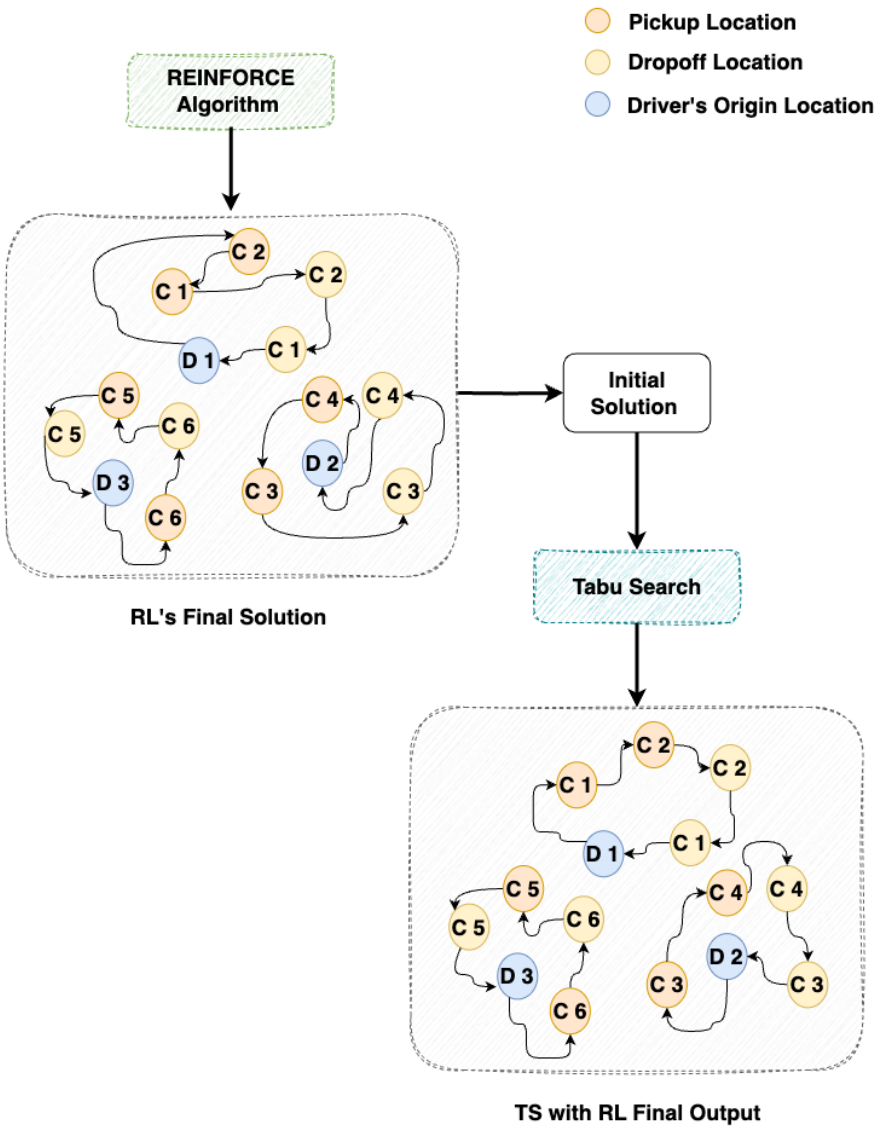


Figure 4.8 Diagram illustrating the data flow of the Tabu Search with Reinforcement Learning experiment

4.6 Software Libraries and Hardware Used

The TS and RL models were implemented using Python 3.9.16. The RL model was built using Tensorflow, a machine-learning library used for training deep neural networks. Table 4.5 shows the packages used for TS and RL implementations and their version.

The parameter optimisation and training of the RL model were run on a Paperspace Linux virtual machine using an 8 virtual-core Intel Xeon Gold 5315Y CPU, 45 GB of RAM, and an NVIDIA RTX A4000 with 16GB of graphics memory.

Table 4.5 Packages used for TS and RL

Packages	**Version**
numpy	1.23.4
tensorflow	2.9.2
keras	2.9.0
optuna	3.4.0
pandas	1.5.0
matplotlib	3.6.1
scipy	1.9.2
plotly	5.18.0
seaborn	0.12.0
tqdm	4.64.1

Once the training of the RL model was complete, the trained model was evaluated on the validation set using a MacBook Air with an 8-core Apple M1 CPU, 8 GB of RAM, and an integrated 7-core GPU. Both TS implementations, one using an equally distributed initial solution and the other using RL solution as the initial solution, were run on this same hardware.

5 Results & Evaluation

This chapter provides the results and evaluation of the three experiments described in the previous chapter. It includes an overview of the objectives defined for this research, an overview of the metrics used for evaluation, and sections for each objective that describe and evaluate each experiment's results.

The first objective of this research involved implementing a TS algorithm as a baseline to solve the MVRRPP and evaluating its performance. The second objective involved modelling the MVRRPP as an RL problem by defining the state, action, and rewards and comparing its performance with the TS algorithm. The third objective involved using the output achieved from the RL algorithm (during the second objective) as the initial solution for the TS algorithm and evaluating its impact on performance when finding the optimal solution for the MVRRPP. The performance of each objective was evaluated using the following five metrics:

1. the waiting time, which is the time it takes for a vehicle to visit a customer's pickup location
2. the travel impact time, which is the difference between the time it takes to drop off a customer from their pickup location and the optimal time from the customer's pickup to the customer's dropoff location without visiting any other customer in between (without ride-pooling)
3. the drivers' distances, which is the total distance travelled in a route by each driver.
4. the running time, which is the total time to solve an MVRRPP instance (a sample in the dataset) when using RL or TS
5. the total number of iterations needed to find the optimal solution when using TS

The travel impact time metric was used to indicate the additional time incurred by the customer due to ride-pooling, from when they are picked up to when they are dropped off. For a quantitative measure, the waiting time and travel impact time metrics represent distance as a base unit, where every 1 unit represents 1 km. The time a customer waits for a vehicle to pick them up is equivalent to the distance the vehicle needs to travel until it reaches the customer's pickup location. The longer the distance a vehicle travels, the longer a customer has to wait for pickup.

Each experiment was evaluated with 4, 5, and 6 drivers, each containing 20, 50, and 100 customers, for a total of 9 datasets. For each group of customers (20, 50, 100), different locations were used for the datasets containing 4, 5 and 6 drivers. Each

dataset consisted of 500 different MVRRPP instances. The following sections will provide the results obtained for each experiment and an evaluation of these results.

5.1 Objective 1 - Tabu Search

For this experiment, the TS algorithm was evaluated on datasets involving different numbers of customers and drivers, with each dataset containing 500 problem instances. TS was evaluated on each instance using an initial solution consisting of an equal distribution of customers across all drivers. This initial solution was used as the starting point for the search process.

Figure 5.1 illustrates an instance from the dataset containing 20 customers and 4 drivers while Figure 5.2 illustrates the final solution of this instance after using TS. The graphs show the locations of each driver's origin location, represented by the green points, and each customer's pickup and dropoff locations, represented as the slate blue and red points, respectively. The final solution of the instance, as depicted in Figure 5.2, shows each driver's route, represented as a line with arrows showing the sequence in which the nodes (locations) were visited by that vehicle. Figure 5.3 highlights each of the four drivers' routes, with each route represented as a different colour. The graphs show how each route starts from its origin location, followed by the customers' pickup and dropoff locations (with the customer's pickup being visited before its respective dropoff), and lastly, ending the route by returning to the origin location.

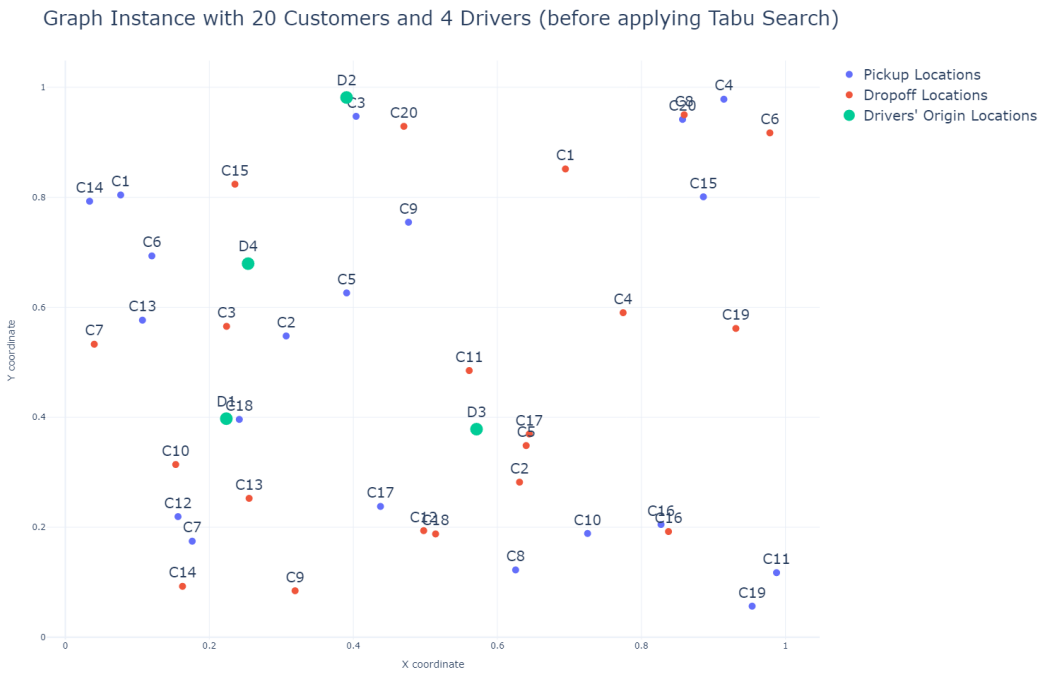


Figure 5.1 Graph Instance with 20 Customers and 4 Drivers before applying TS

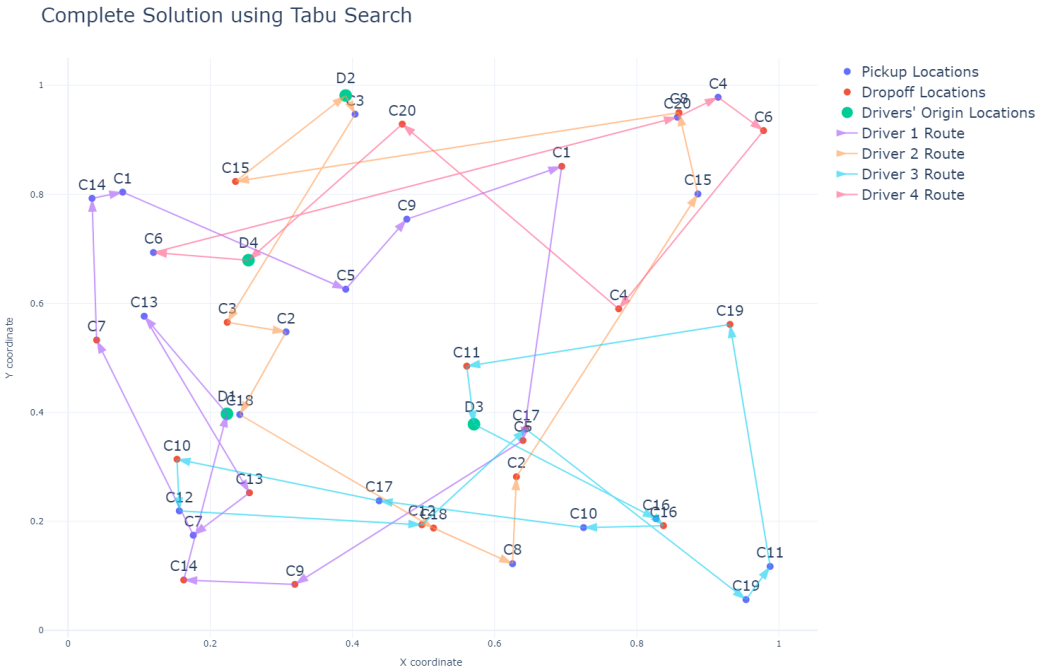


Figure 5.2 Final Solution after applying TS on the graph instance of 20 customers and 4 drivers

(a) Driver 1 Route

(b) Driver 2 Route

(c) Driver 3 Route

(d) Driver 4 Route

Figure 5.3 Drivers' Routes obtained using TS on an instance with 4 drivers and 20 customers

Figures 5.4 to 5.8 show the spread and distribution of the data. Each box plot shows a box representing the interquartile range that indicates where the middle 50% of the data lies. The box's lower and upper edges represent the first and third quartiles (with each quartile representing 25% of the data). The horizontal line inside the box represents the median of the data. The interquartile range indicates how the samples in the data are dispersed. The lines extending vertically on each box from the quartiles represent the 'whiskers' and indicate the data's minimum (lower line) and maximum (upper line) values. These whiskers provide insight into the range of the data. The data points outside the whiskers indicate the outliers within the data.

Figure 5.4 shows a box plot with the waiting times across 500 samples for each driver instance, consisting of 20, 50, and 100 customers. As the number of drivers increases with the same number of customers, both the minimum and maximum range of waiting times tend to decrease, suggesting a reduction in the spread of waiting times as more customers are distributed amongst drivers, reducing the overall waiting time as more customers can be attended to by different drivers. The interquartile range also becomes slightly narrower, suggesting that the variability in waiting times diminishes as it is more concentrated around the median. However, when comparing the waiting time across the same number of drivers but with additional customers, the interquartile range increases, indicating a higher variability along the centre of the box plot among each group of customers. The minimum and maximum range of waiting times also increases with the increase in customers, indicating a broader spread of waiting times across different customers. This broader spread indicates the increasing complexity and challenges when finding optimal routes with more customers, as additional locations must be visited, increasing the overall waiting time.

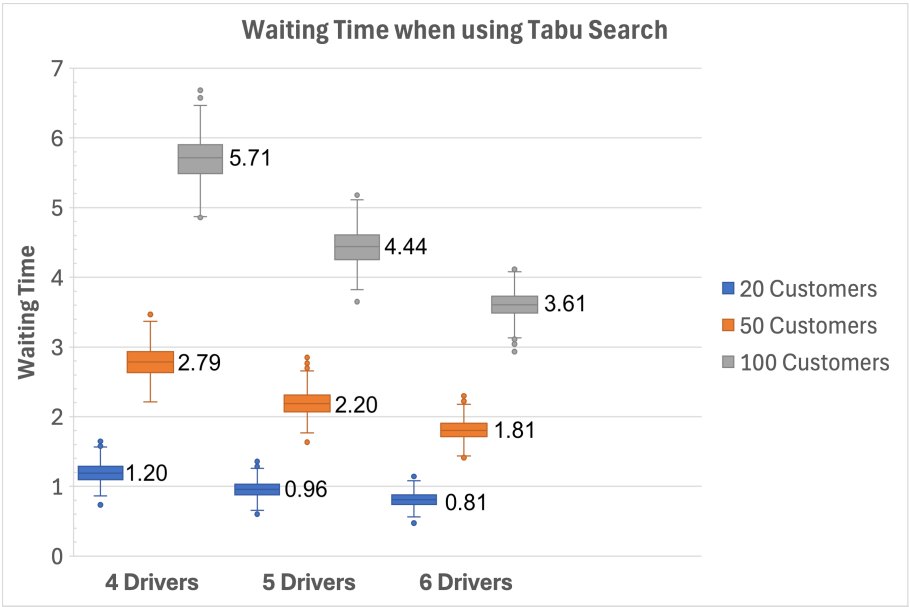


Figure 5.4 Box Plot with Waiting Time across different instances when using TS

Figure 5.5 illustrates a box plot showing the travel impact time on different driver instances with various customer groups. For each box plot belonging to the same number of customers, the interquartile range narrows slightly as more drivers are added, suggesting less variation in the travel impact time for the middle (50%) of the data as the drivers increase. Each box plot's minimum and maximum range also reduces as the drivers increase within the same customer group, indicating a smaller spread between the shortest and longest travel impact time in the data as the drivers increase. This shows that as the number of drivers increases, the travel impact time is reduced as the customers' demand is being shared across more drivers' routes. However, when comparing the box plot of 6 drivers with that of 5 drivers in instances of 100 customers, there was a slight increase in the box plot's interquartile range and the minimum and maximum range. This suggests a small increase in variability in the travel impact time on the box plot with 6 drivers. The box plots also indicate an increase in the median travel time as more customers are added to an instance of the same number of drivers. This is likely due to the driver making more stops along the route to pick up or drop off additional customers, increasing the time a customer travels between pickup and destination.

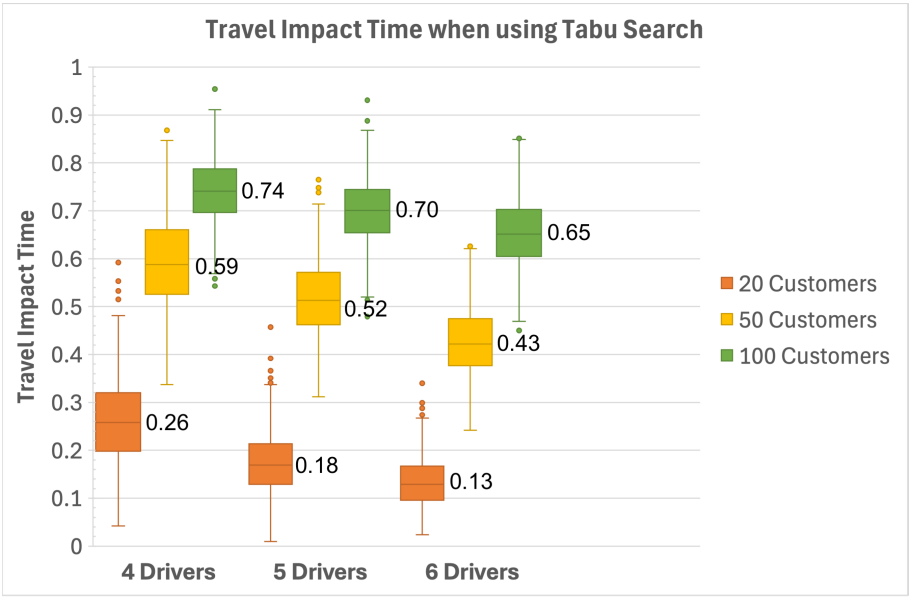


Figure 5.5 Box Plot with Travel Impact Time across different instances when using TS

A box plot illustrating the driver distances in instances with different drivers and customers is shown in Figure 5.6. The box plot shows that the interquartile range and minimum and maximum range decrease as the number of drivers increases in instances of the same number of customers. The decrease in the minimum and maximum range shows a reduction in the spread of distances, whilst the decrease in the interquartile range shows that the variability within the central portion was reduced. As the number of drivers increases, the average total distance decreases as more customers are being

picked up by different vehicles, decreasing the number of stops a vehicle must make. The box plots also indicate that as customers increased within instances of the same number of drivers, the median values of the box plots also increased, along with the interquartile range and minimum range. This shows how the distance travelled by the drivers increases as the number of customers increases, as the driver has to make additional stops along the route for further pickups and dropoffs. Each additional stop adds distance to the overall route as the vehicle must travel from one stop to the next.

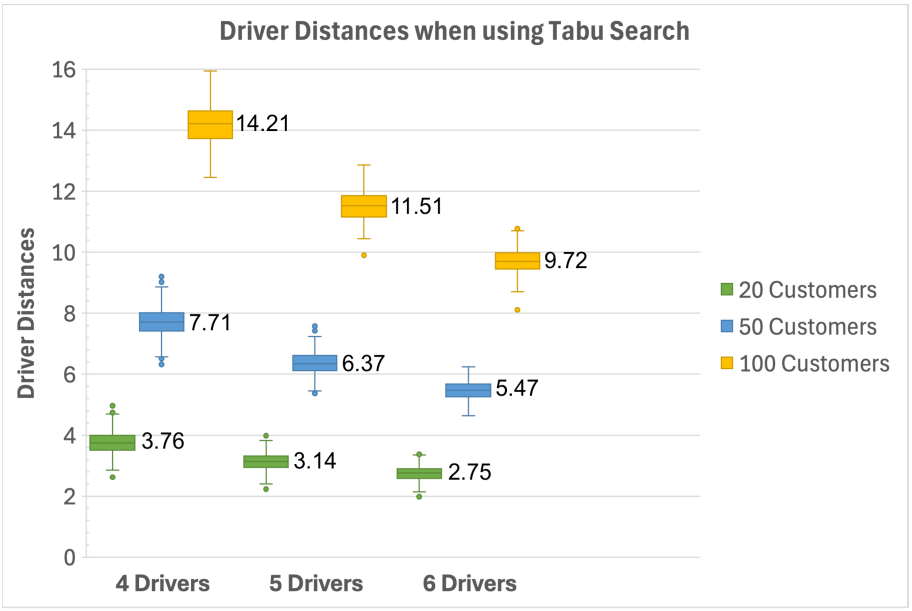


Figure 5.6 Box Plot with Drivers' Distances across different instances when using TS

Figure 5.7 illustrates a box plot with the number of iterations TS took to generate solutions across varying numbers of drivers and customers. The box plot shows that the median values and the size of the box plots remain consistent across different numbers of drivers with the same customers. As the number of customers increased within instances of the same number of drivers, the min-max range and median also increased. This indicates that the solution space was becoming more complex and challenging and required more iterations to converge.

Figure 5.8 shows box plots with the running time taken to generate solutions using TS for each group of customers across multiple drivers. Within instances of the same number of customers, the running time increased slightly as the number of drivers increased; this is most noticeable in instances with 50 and 100 customers. The slight increase in running time between drivers indicates that the complexity of the problem expanded as more driver routes needed to be considered, requiring the TS to take longer computation times to find the local optimal solution. The box plots also indicate a substantial increase in running time when adding additional customers, as indicated by the box plot's minimum and maximum range and median, suggesting that the TS struggles to handle large instances within a reasonable time frame.

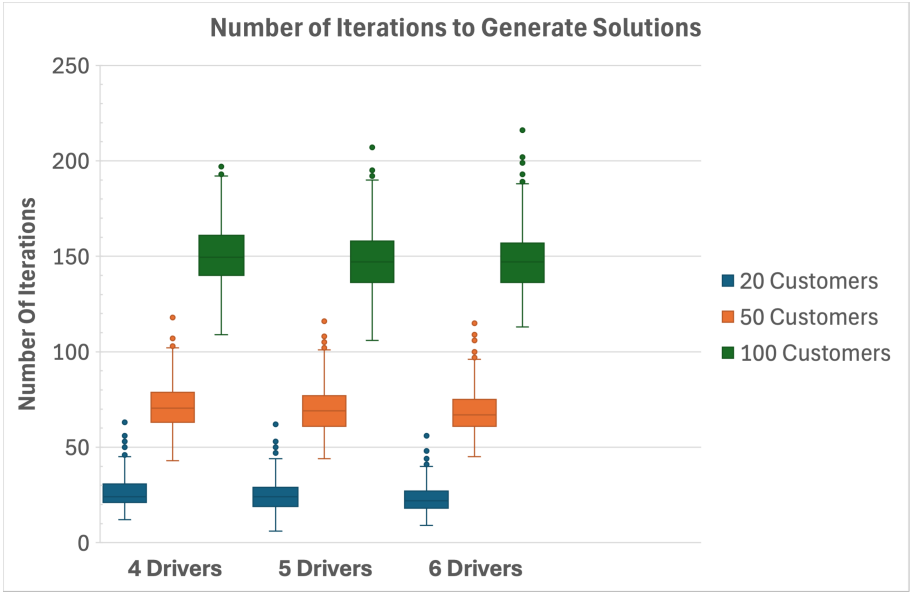
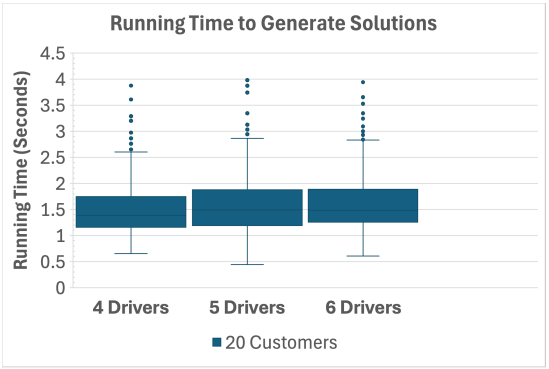
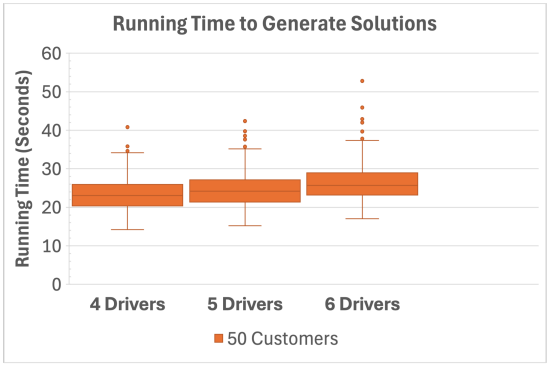


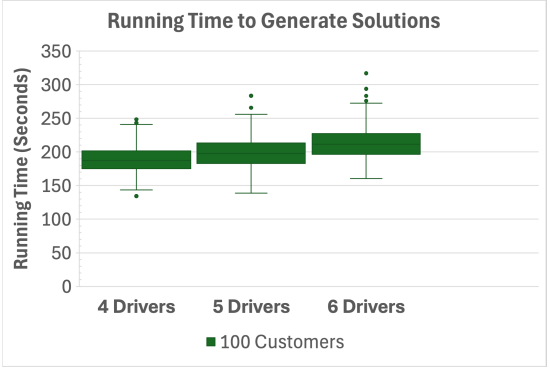
Figure 5.7 Number of Iterations to generate solutions when using TS



(a) 20 Customers



(b) 50 Customers



(c) 100 Customers

Figure 5.8 Running Time to generate solutions when using TS

Figures 5.9 to 5.11 illustrate the relationship between the number of iterations and the time taken to find a solution for each instance in the dataset. Each scatter plot represents instances of 20, 50, and 100 customers, showing the number of iterations and the time TS takes to find the 'best solution' for each instance in the datasets containing 4, 5 and 6 drivers. The scatter plots show the algorithm's complexity; as the number of drivers and customers increases, the number of possible solutions increases, and the search space and neighbourhood becomes more larger. Therefore, converging to a local optimum takes more iterations, leading to increased computational time to find a solution, as apparent in each scatter plot with the increase in drivers.

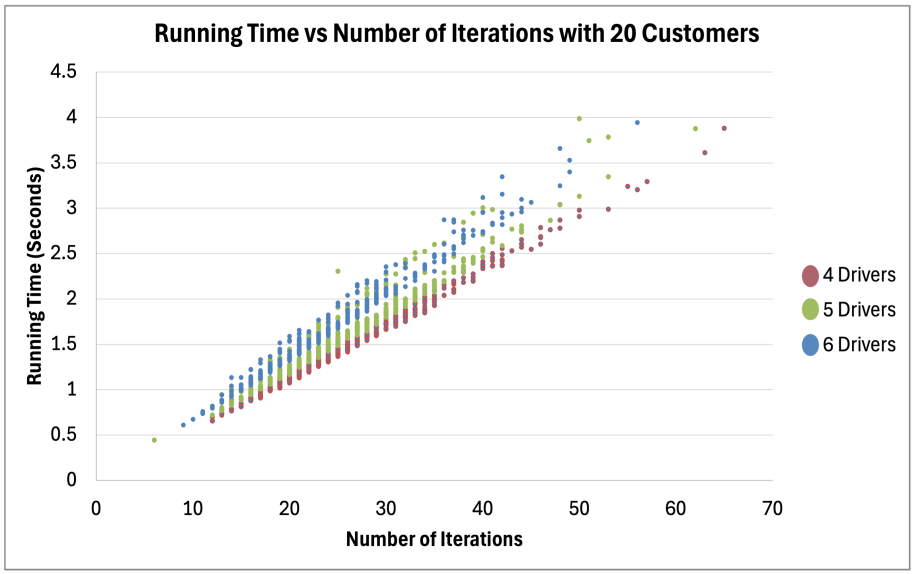


Figure 5.9 Running Time vs Number of Iterations when using TS with 20 Customers

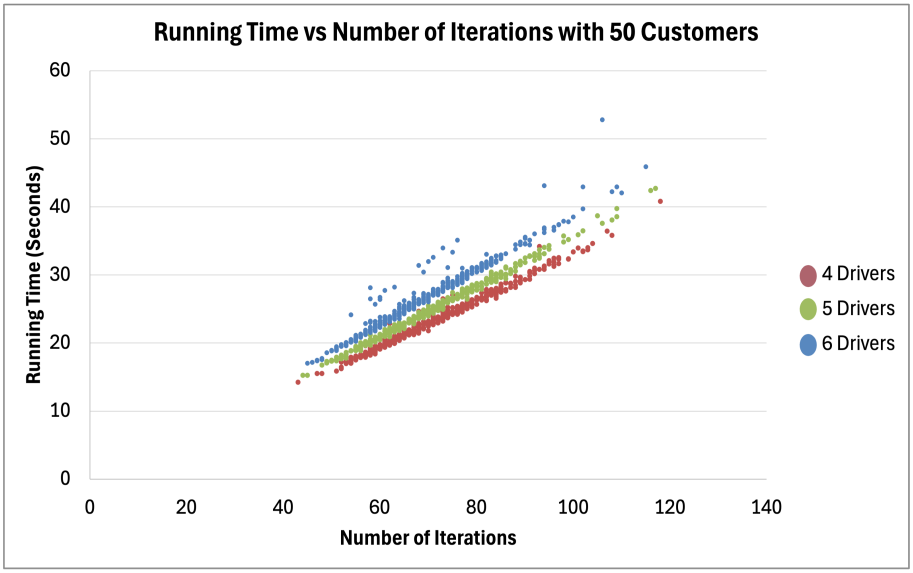


Figure 5.10 Running Time vs Number of Iterations when using TS with 50 Customers

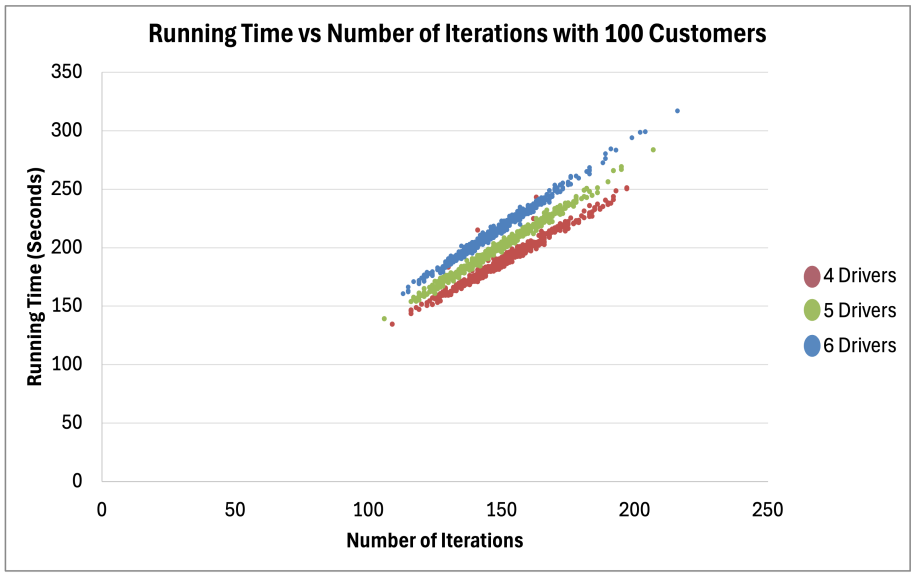


Figure 5.11 Running Time vs Number of Iterations when using TS with 100 Customers

5.2 Objective 2 - The Reinforcement Learning Model

The second experiment involved evaluating the performance of the trained RL models across instances with different numbers of customers and drivers. For comparison purposes, the RL models were evaluated using the same dataset that was used to evaluate the TS algorithm.

Figure 5.12 displays a graph depicting an instance from the dataset consisting of 20 customers and 4 drivers, while Figure 5.13 shows the final solution of this instance after using RL. Figure 5.14 illustrates each driver's route as separate graphs.

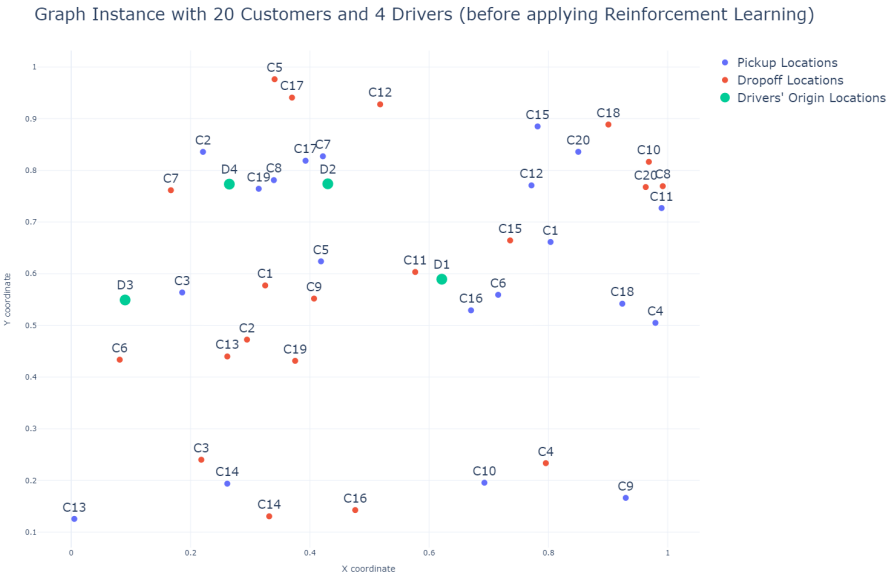


Figure 5.12 Graph Instance with 20 Customers and 4 Drivers before applying RL

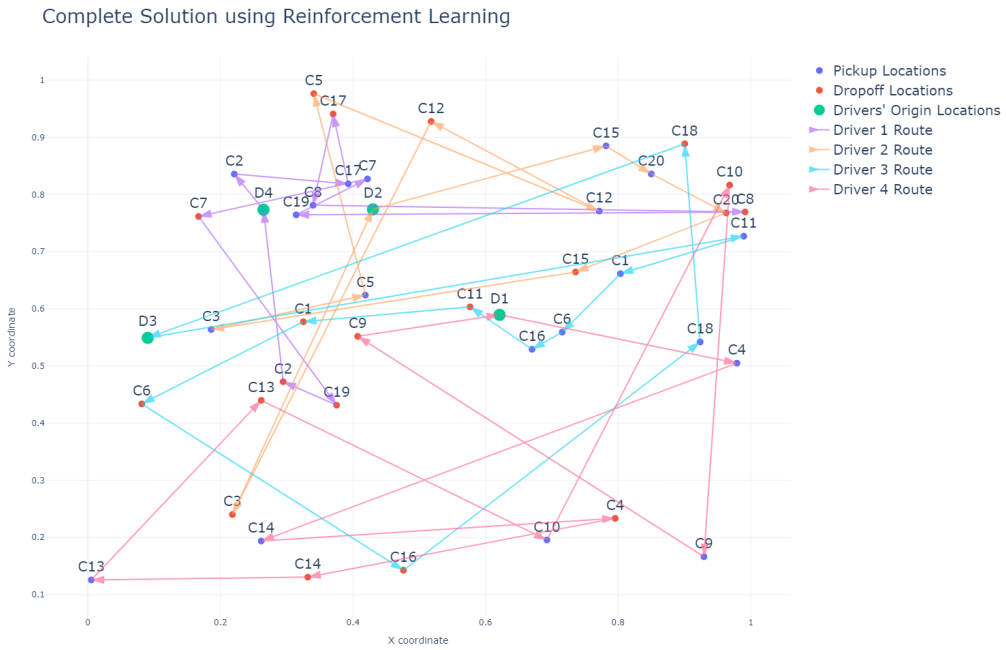
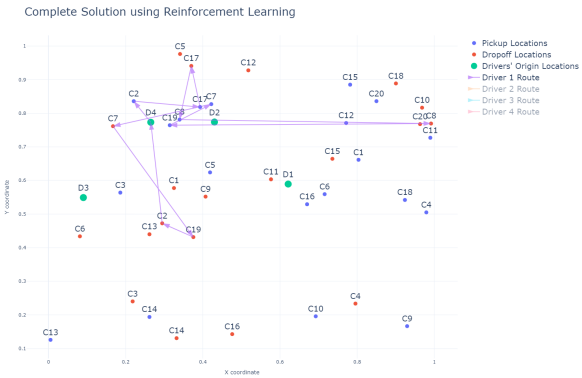
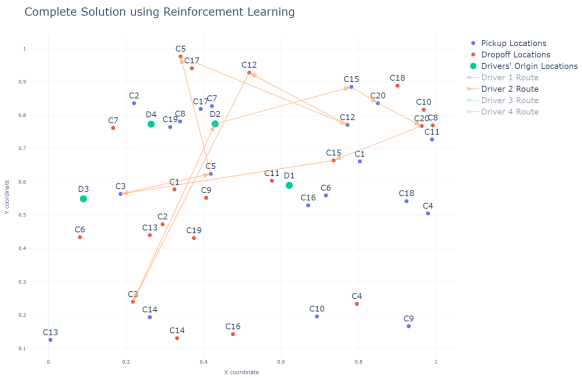


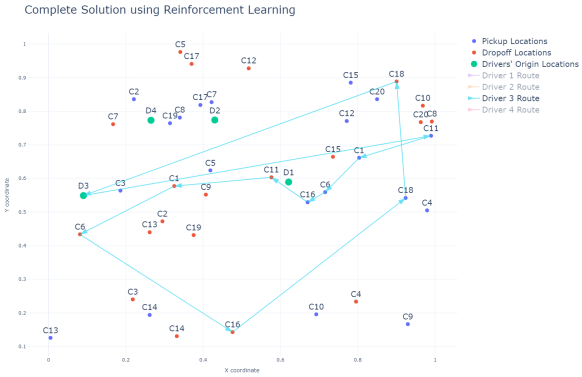
Figure 5.13 Final Solution after applying RL on the graph instance of 20 customers and 4 drivers



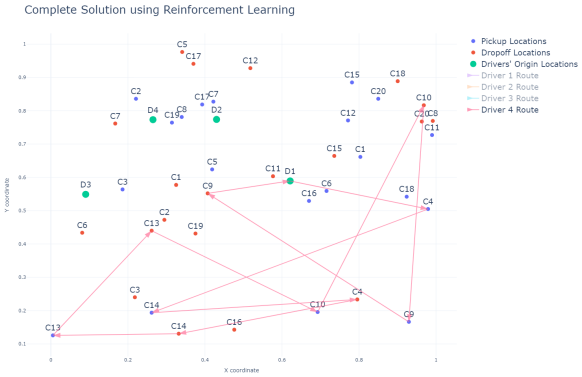
(a) Driver 1 Route



(b) Driver 2 Route



(c) Driver 3 Route



(d) Driver 4 Route

Figure 5.14 Drivers' Routes obtained using RL on an instance with 4 drivers and 20 customers

Figure 5.15 illustrates a bar chart with the average customer's waiting time when using RL across different instances of customers and drivers. The bar charts show that the average waiting time increases as the number of customers increases in instances with the same number of drivers. This is noticeable in instances with 100 customers as the waiting time significantly increases. As the number of customers increases, the driver has to visit more locations for pickup and drop off, increasing the overall waiting time of customers. When comparing the performance across different numbers of drivers with the same amount of customers, the waiting time varies and appears to be non-linear, with only a few instances indicating a linear improvement in waiting time as the number of drivers increases.

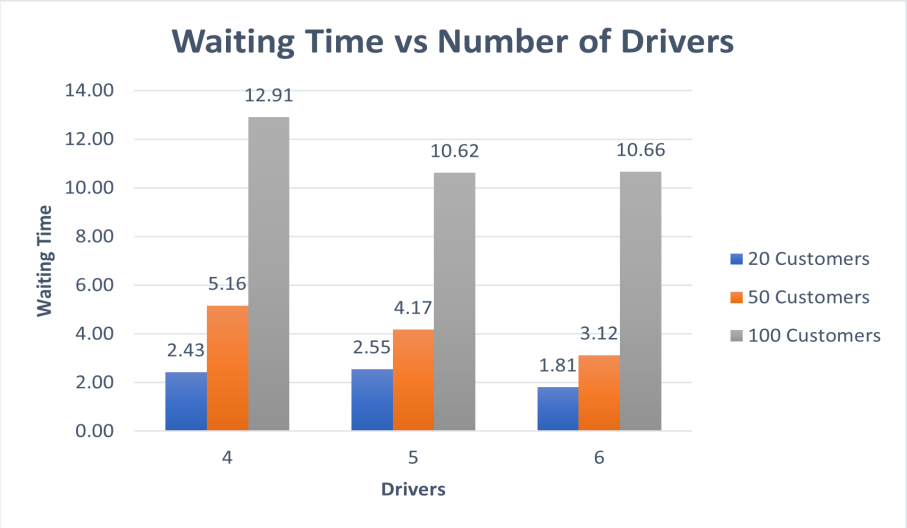


Figure 5.15 Average Waiting Time vs Number of Drivers when using RL

Figure 5.16 illustrates a box plot displaying the waiting time performance when using RL across all instances. The box plot shows that as customers increase, the box plot position moves further up, showcasing an increase in waiting time.

Figure 5.17 shows a bar chart with the average travel impact time across different RL instances. The bar chart indicates that the travel impact time increases as more customers are added to a driver instance. When comparing instances between 4 and 5 drivers, the travel impact time across each customer group decreased. However, when comparing instances between 5 and 6 drivers, the travel impact time increased in instances of 20 and 50 customers, whilst in instances with 100 customers, the average travel impact time remained the same, suggesting that the additional driver did not improve performance.

Figure 5.18 shows a box plot with the travel impact time performance across all instances using RL. The box plot shows a similar performance between 20 and 50 customers, with the travel impact time increasing exponentially with instances of 100 customers.

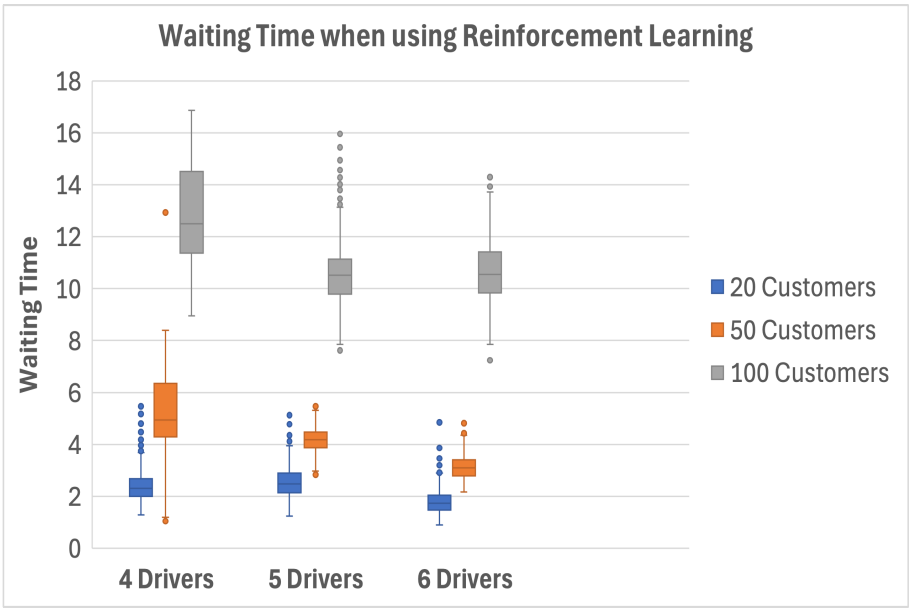


Figure 5.16 Box Plot with Waiting Time across different instances of drivers and customers using Reinforcement Learning

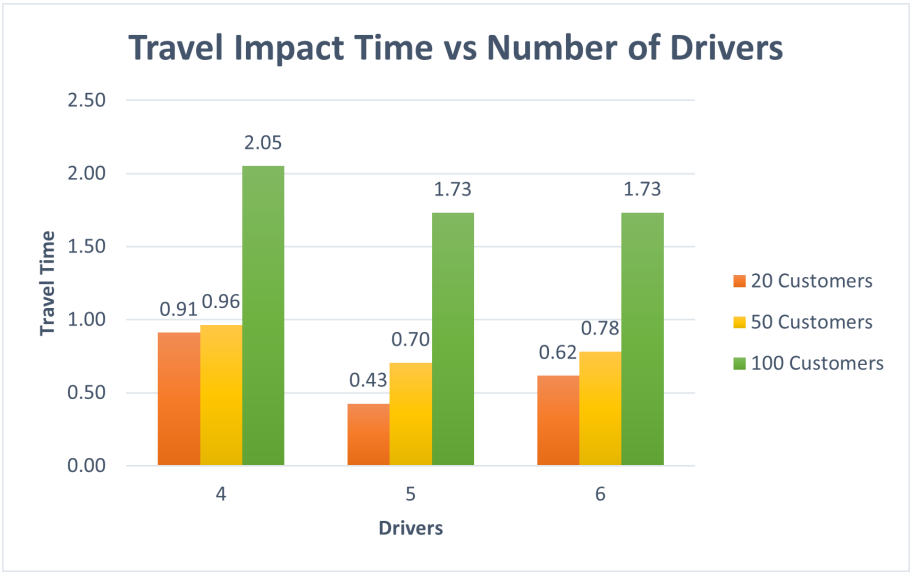


Figure 5.17 Average Travel Impact Time vs Number of Drivers when using RL

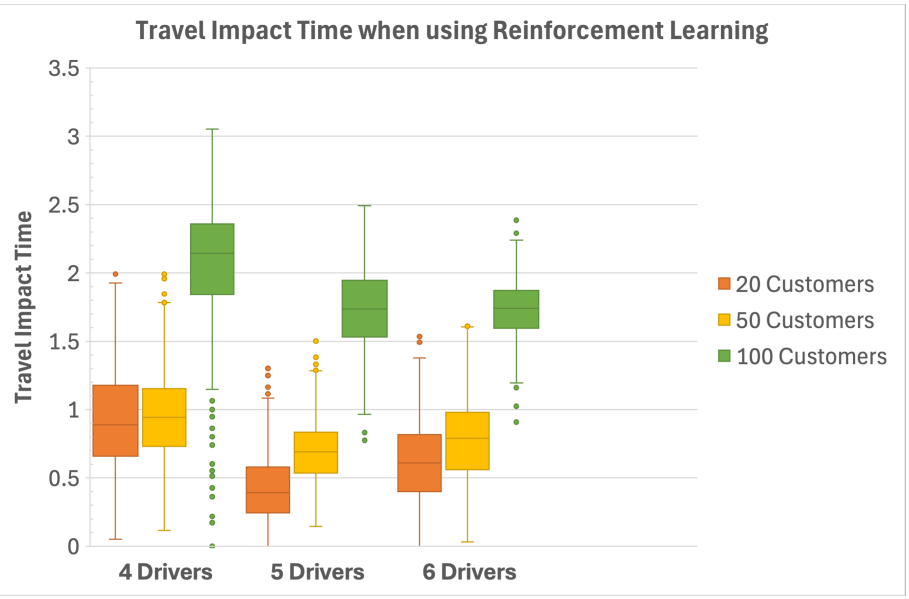


Figure 5.18 Box Plot with Travel Impact Time across different instances of drivers and customers using Reinforcement Learning

The increase and minimal impact in waiting time and travel impact time on certain instances when adding more drivers indicate RL's challenges in finding the trade-off between waiting time and travel impact time. As the number of drivers increases, the RL model attempts to balance the allocation of customers among these drivers to minimise overall waiting time and travel impact time. However, with increased drivers, RL may distribute customers among drivers less efficiently, resulting in longer times. The RL model seeks to find the balance where waiting time is minimised without increasing the customer's travel impact time.

The bar chart presented in Figure 5.19 shows the average total distance travelled by the drivers in different instances using the trained RL model. For each driver, the distance travelled increased as more customers were added. In contrast, the bar chart shows that the distances decreased as more drivers were added to instances with the same number of customers. The decrease in distance as more drivers are added to an instance suggests that the RL model is effectively optimising the order in which customers are visited by assigning pickups and drop-offs based on proximity and effectively distributing customers across routes to minimise the overall distance travelled.

Figure 5.20 exhibits a box plot showing the distance performance when using RL. The box plot confirms that the distances increase as the number of customers increases across all driver instances. It also confirms that the distances decrease across instances with the same number of customers when adding more drivers.

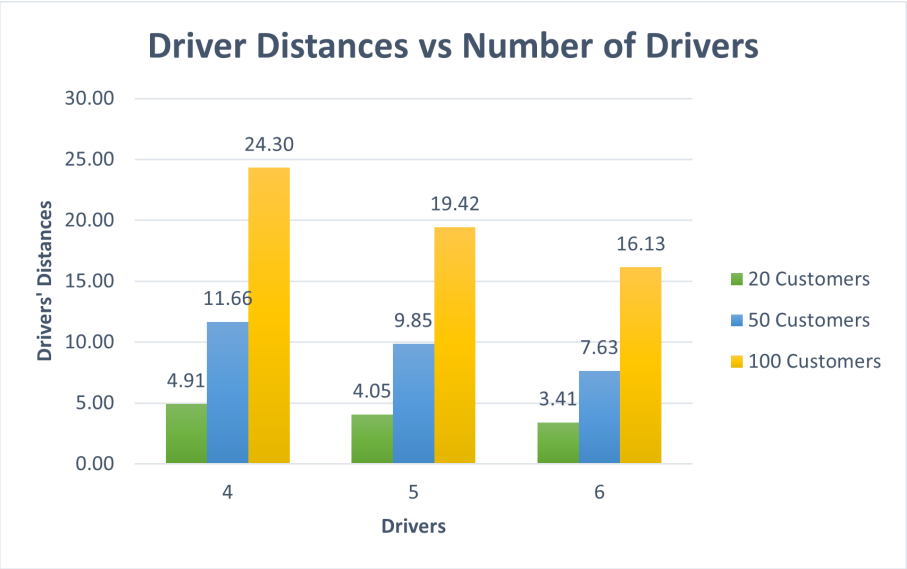


Figure 5.19 Average Driver Distances vs Number of Drivers when using RL

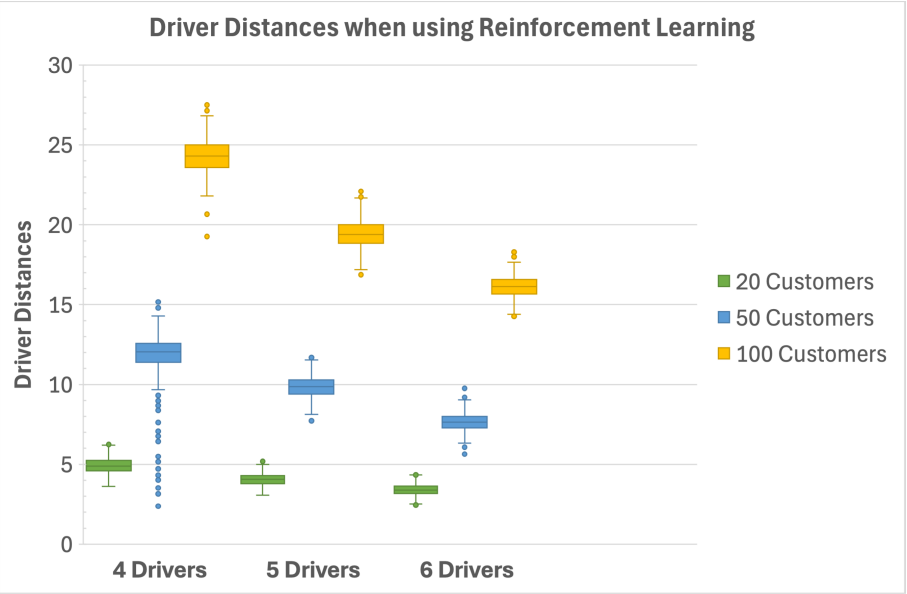


Figure 5.20 Box Plot with the drivers' distances across different instances of drivers and customers using Reinforcement Learning

The bar chart in Figure 5.21 shows the average running time for the RL to complete each solution. The running time increases as additional customers are added and remains consistent with the addition of drivers. The increase in running time shows how, as the number of customers increases, the problem complexity grows exponentially, and the RL model needs additional time to consider more possible combinations.

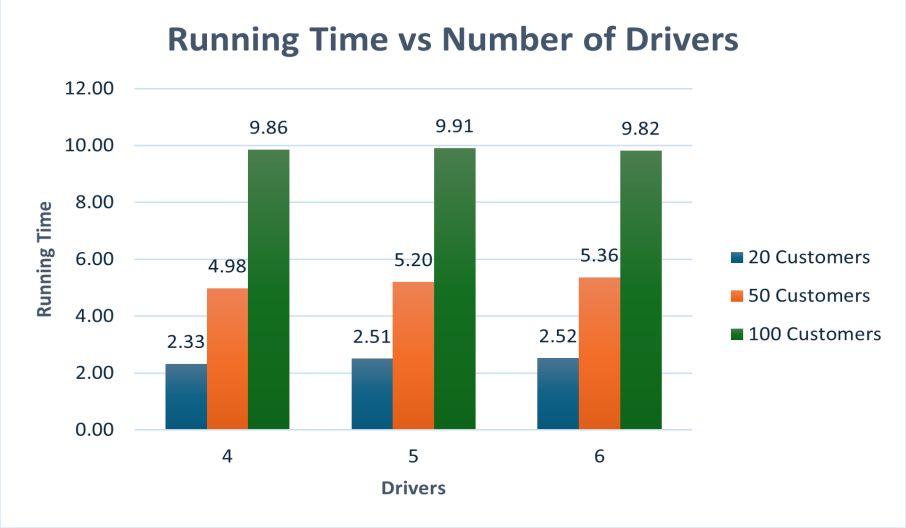


Figure 5.21 Average Running Time vs Number of Drivers when using RL

The box plot in Figure 5.22 shows RL's running time performance across each instance. It indicates that it takes longer to complete solutions as the number of customers increases; however, the running time remains consistent as the number of drivers increases across instances with the same number of customers.

Figure 5.22 Box Plot with the Running Time across different instances of drivers and customers using Reinforcement Learning

Figures 5.23 to 5.25 show the total average waiting time for both TS and RL experiments on instances with 20, 50 and 100 customers across different number of drivers. Lower bars indicate a better performance as they represent shorter waiting times. The bar charts show that the TS performed better in terms of waiting time for each customer group across each driver instance.

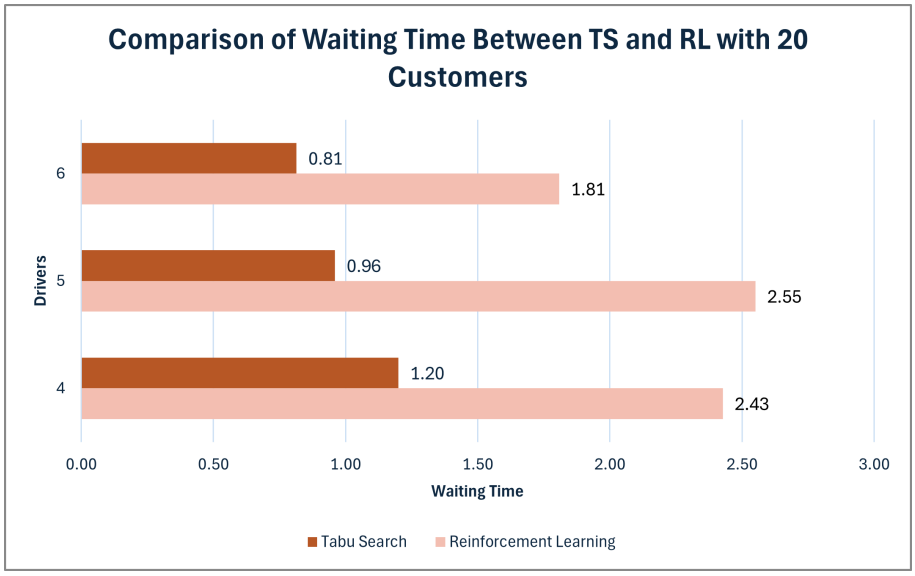


Figure 5.23 Comparing Average Waiting Time between TS and RL with 20 customers across each driver instance

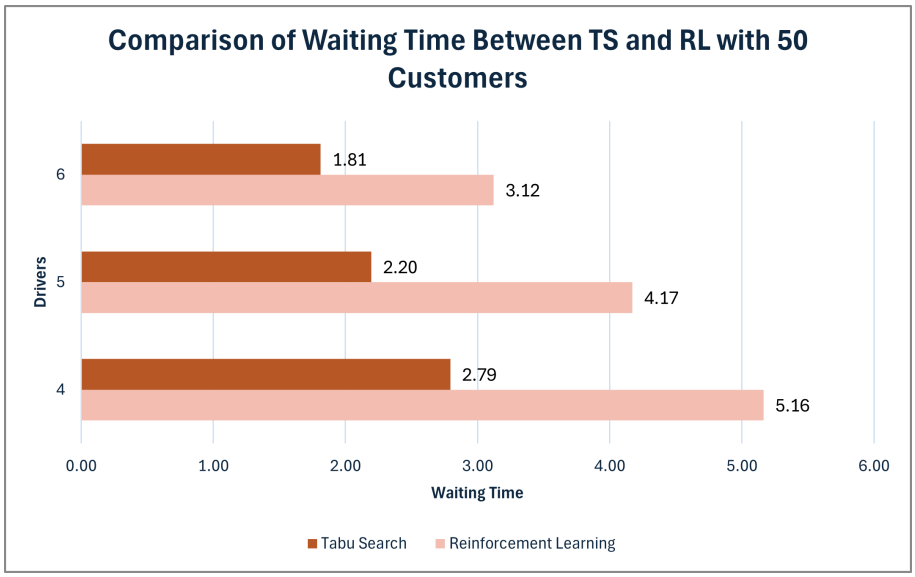


Figure 5.24 Comparing Average Waiting Time between TS and RL with 50 customers across each driver instance

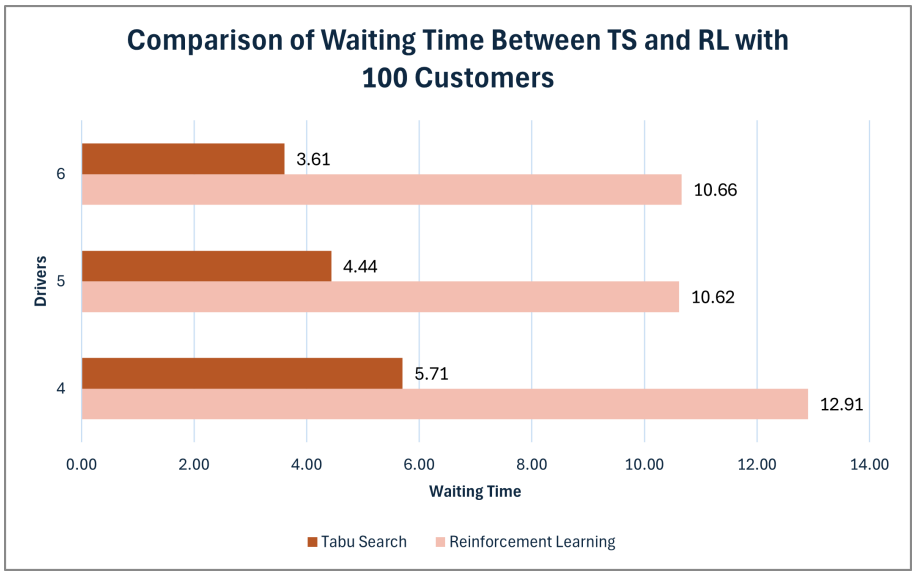


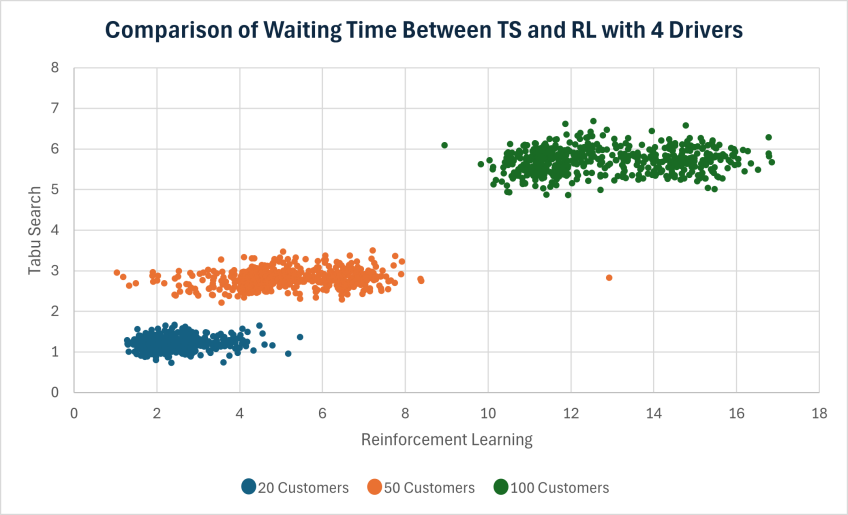
Figure 5.25 Comparing Average Waiting Time between TS and RL with 100 customers across each driver instance

Figure 5.26 shows the scatter plots comparing the waiting time performance of TS and RL for each sample on the same dataset with the respective 4, 5 and 6 drivers. The scatter plots show a similar performance across each driver instance.

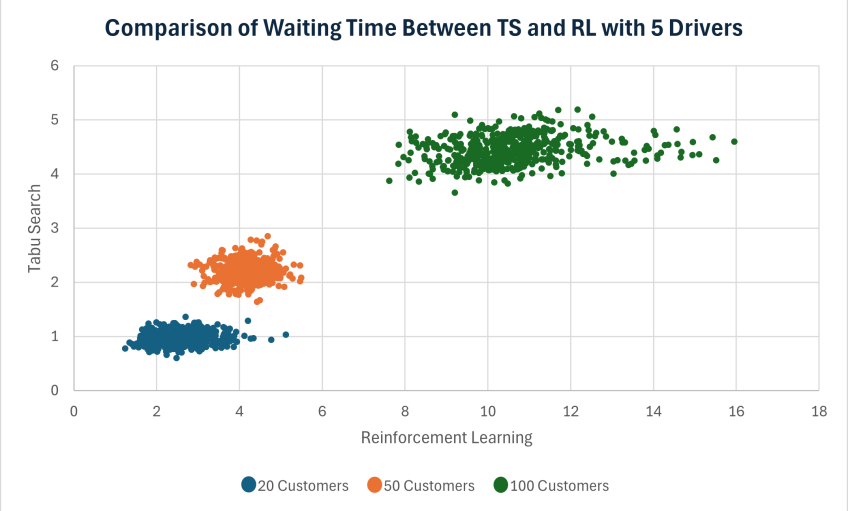
When comparing instances of 20 customers, the data points on each scatter plot show a cluster of data located close to the x-axis and y-axis, indicating low waiting times for both TS and RL. The waiting times are slightly more spread along the x-axis, suggesting a wider range in values when using RL.

When comparing scenarios with 50 customers, the instance with 4 drivers showed a wider spread of data points along the RL values compared to instances with 5 and 6 drivers, with less spread and more tightly packed data points. Across all driver instances with 50 customers, TS values were overall lower in terms of waiting time than RL.

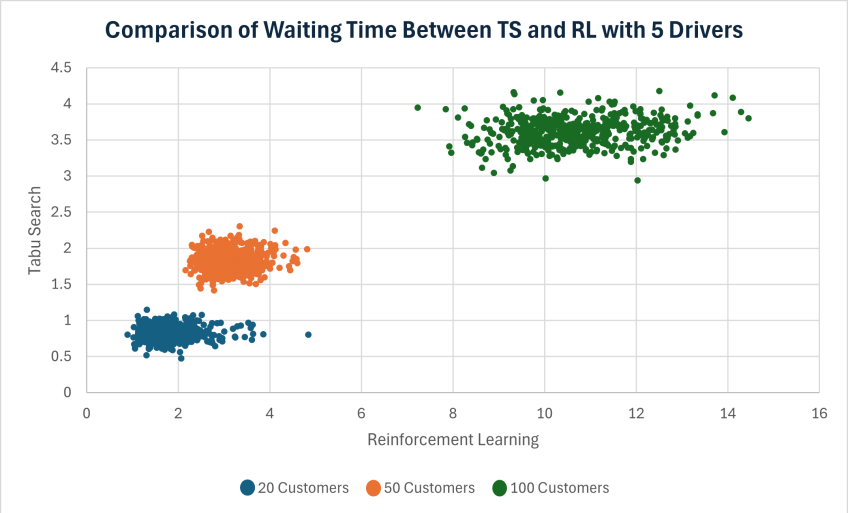
The scatter plots showed that for each driver with 100 customers, the waiting time values were spread mainly across the x-axis (RL values) and contained higher values compared to the y-axis (TS values). This showed that TS performed better with lower waiting times and less variance between samples compared to RL.



(a) 4 Drivers



(b) 5 Drivers



(c) 6 Drivers

Figure 5.26 Scatter plot comparing Waiting Time Between TS and RL

Figures 5.27 to 5.29 illustrate the average travel impact time for TS and RL across different driver instances with 20, 50, and 100 customers. The bar charts indicate that TS performed better with lower overall travel impact time values across each scenario.

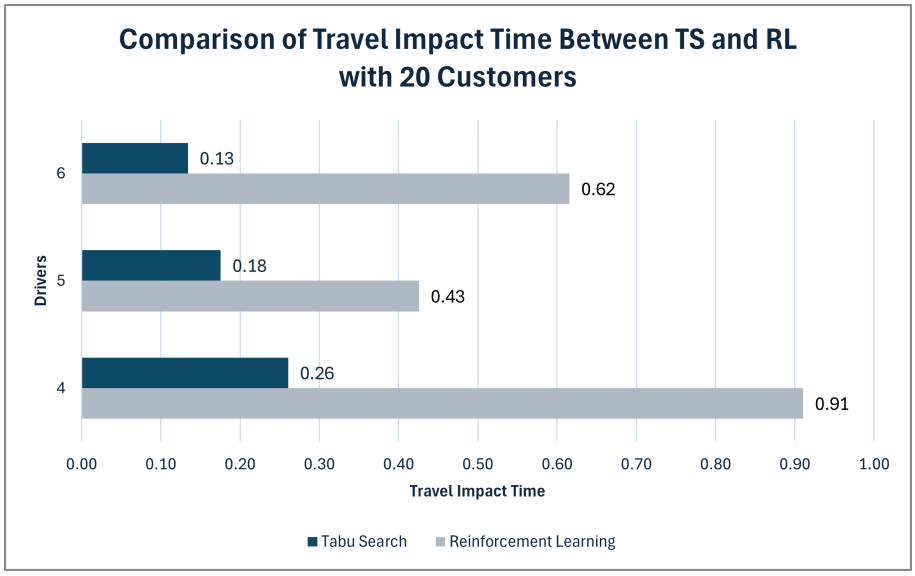


Figure 5.27 Comparing Average Travel Impact Time between TS and RL with 20 customers across each driver instance

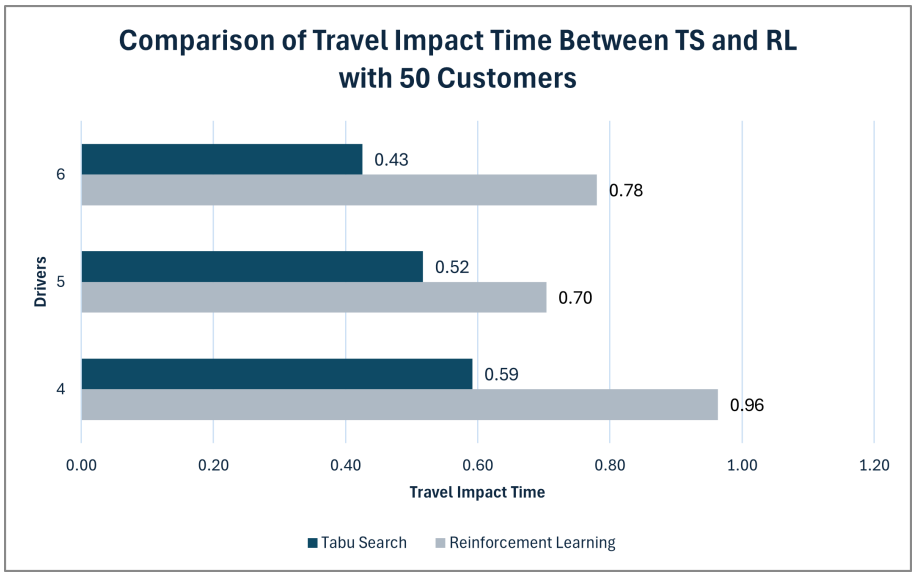


Figure 5.28 Comparing Average Travel Impact Time between TS and RL with 50 customers across each driver instance

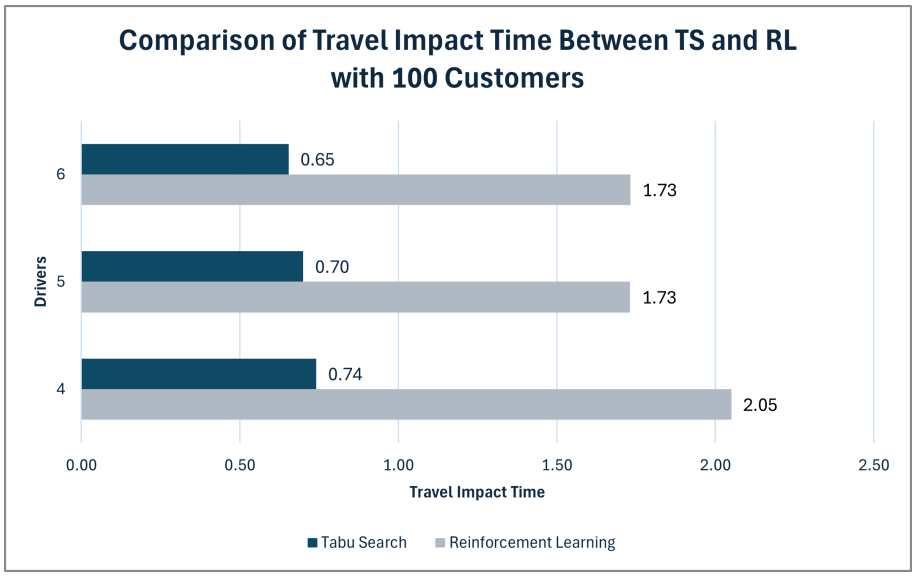
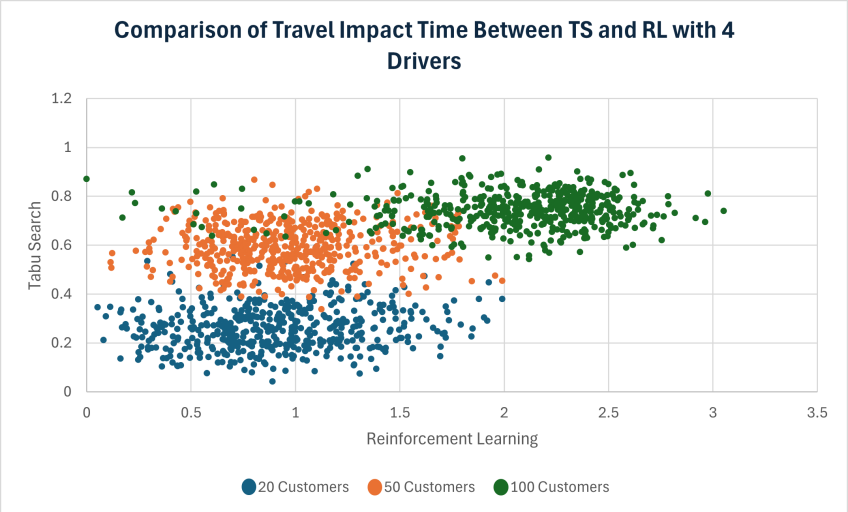


Figure 5.29 Comparing Average Travel Impact Time between TS and RL with 100 customers across each driver instance

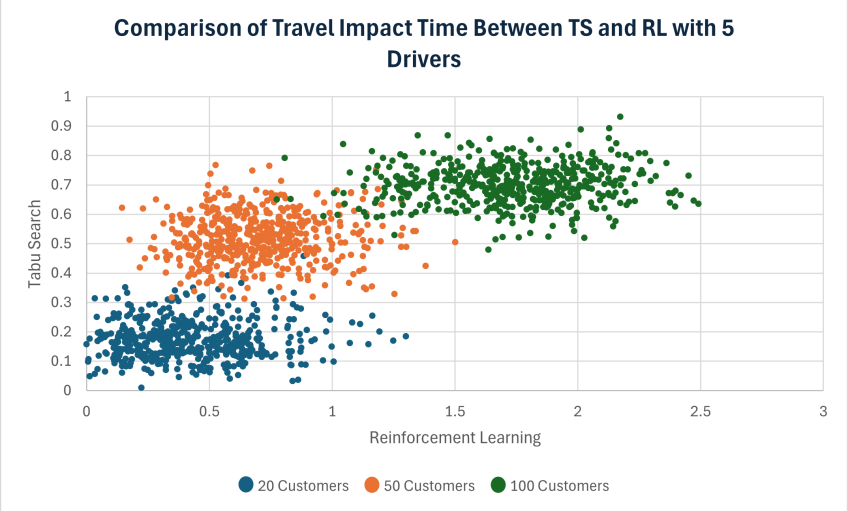
The scatter plots in Figure 5.30 display the travel impact time performance for both TS and RL experiments using different customers and drivers. Both models were compared on the same datasets.

Each group of customers showed a wide spread of data points across the x-axis (RL values) with higher values along the same axis. This indicates that the RL had a larger variation and higher travel impact times than TS.

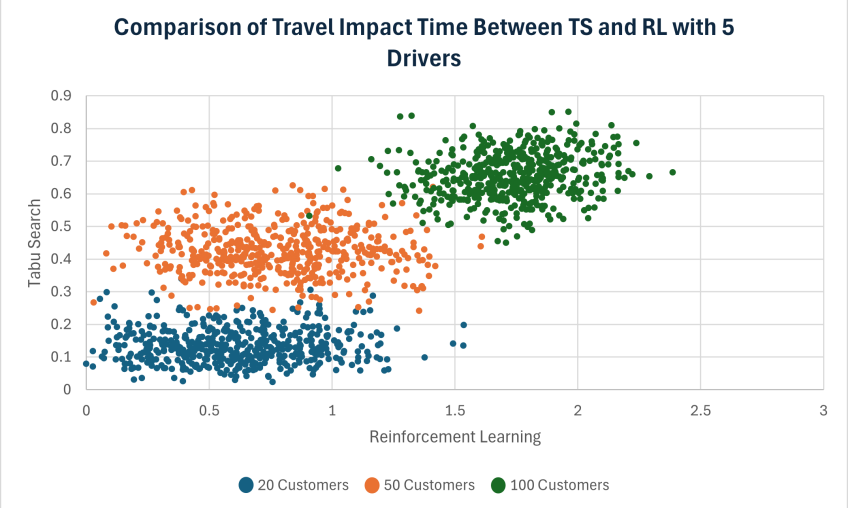
As the number of customers increased across all drivers, the data points for each customer group moved slightly further up, indicating a slight increase in travel impact time when using TS. However, the travel impact times were still significantly lower than RL. When comparing the travel impact time across multiple customer groups, it is proven that the customers spent less time between pickup and dropoff when using TS.



(a) 4 Drivers



(b) 5 Drivers



(c) 6 Drivers

Figure 5.30 Scatter Plot comparing Travel Impact Time Between TS and RL

The average drivers' distances of the TS and RL, as illustrated in Figures 5.31 to 5.33, show the performance of the two models across instances of 20, 50 and 100 customers with different numbers of drivers. The bar charts show that TS had the least distance travelled across all scenarios compared to RL, highlighting its effectiveness at optimising the routes to reduce the overall distance.

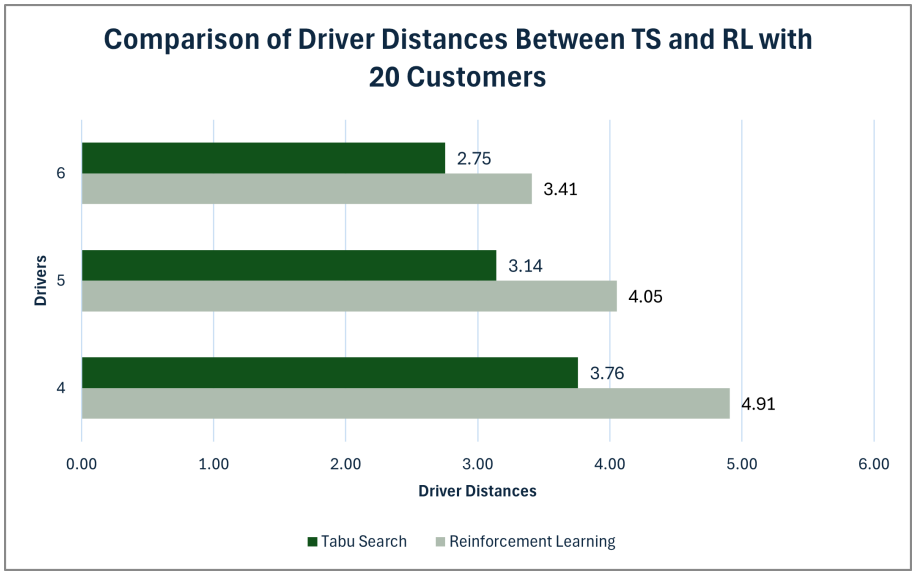


Figure 5.31 Comparing Average Distance Travelled between TS and RL with 20 customers across each driver instance

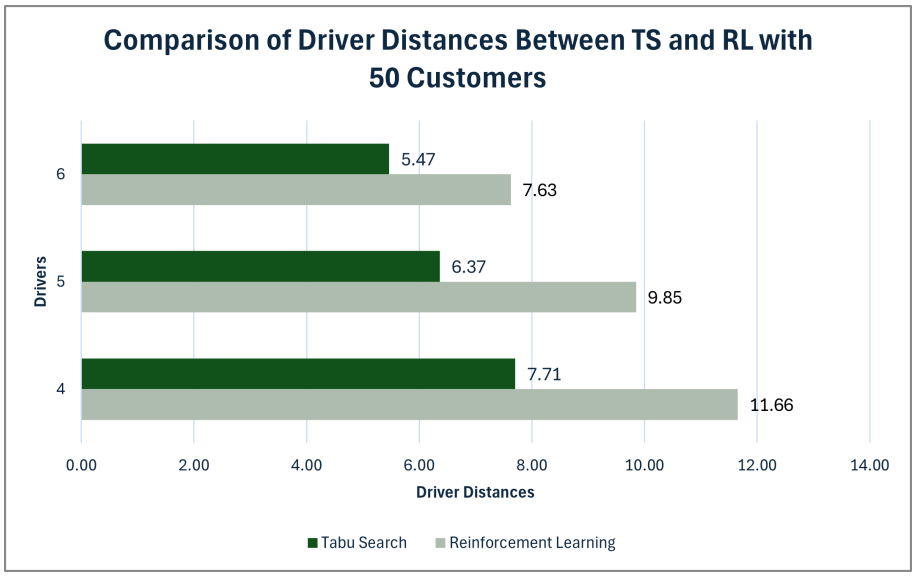


Figure 5.32 Comparing Average Distance Travelled between TS and RL with 50 customers across each driver instance

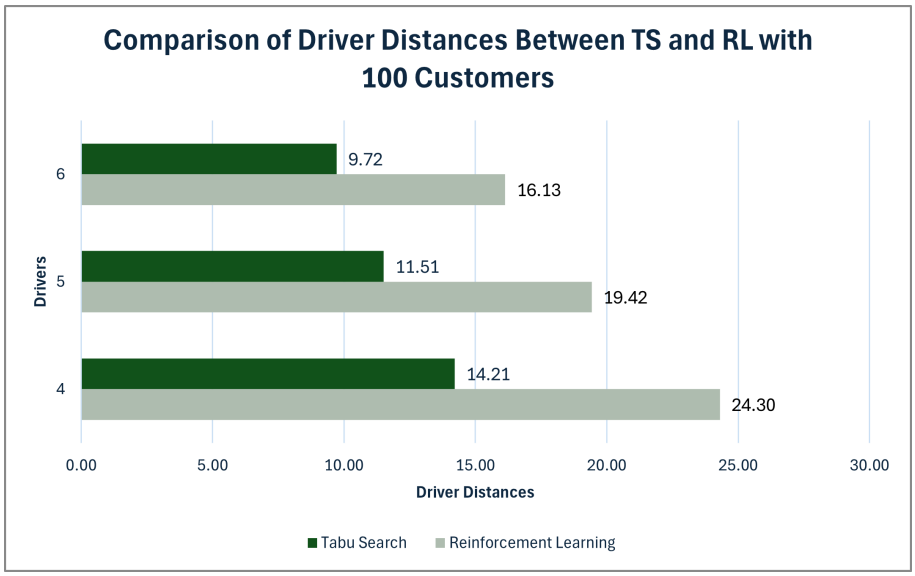
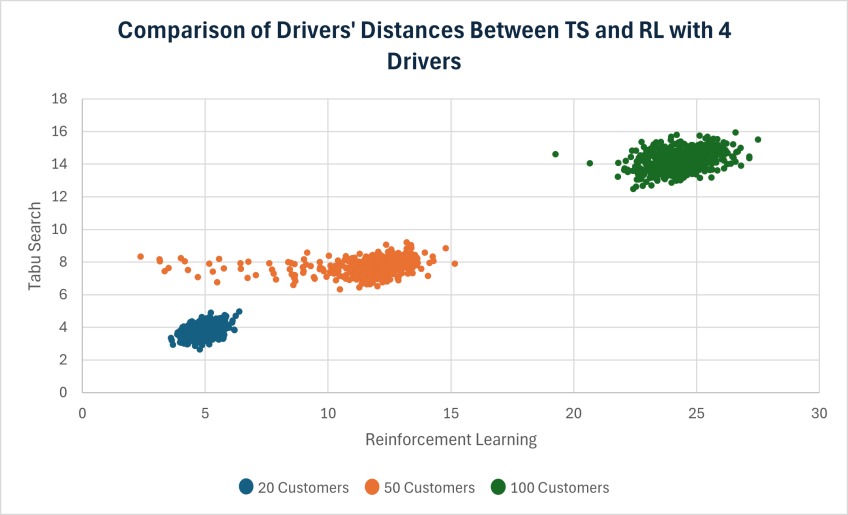


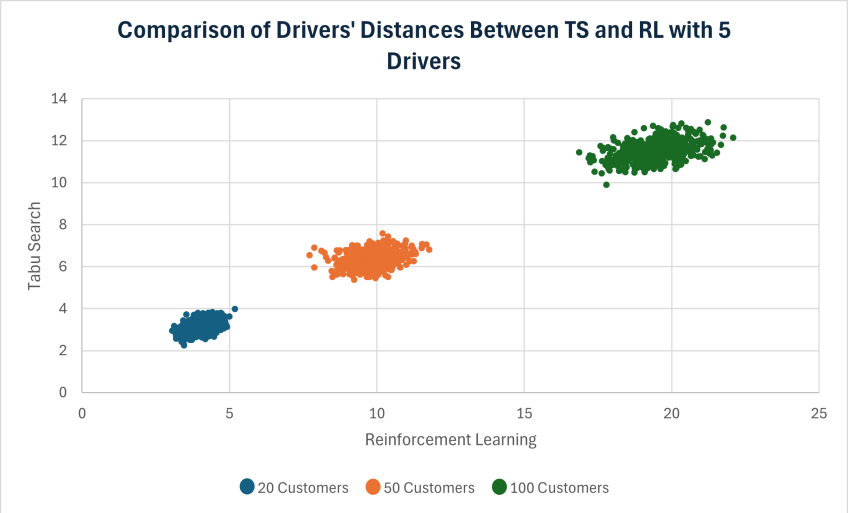
Figure 5.33 Comparing Average Distance Travelled between TS and RL with 100 customers across each driver instance

Figure 5.34 shows the performance of the TS and RL when comparing the total distance travelled on each sample across instances of 4, 5 and 6 drivers with multiple groups of customers. The scatter plots show clusters of data points for each customer group across all drivers. The clusters indicate that as the number of customers increased, both models saw an increase in distance travelled. The scatter plots also show that as the number of customers in an instance increases, the margin in the distance travelled values between the two models increases, with RL having higher distances than TS. The scatter plots also showed a slight increase in the spread of data points as the number of customers increased. In the instance of 50 customers with 4 drivers, an anomaly increased the number of outliers along the RL x-axis, causing an increase in the variation of distance travelled.

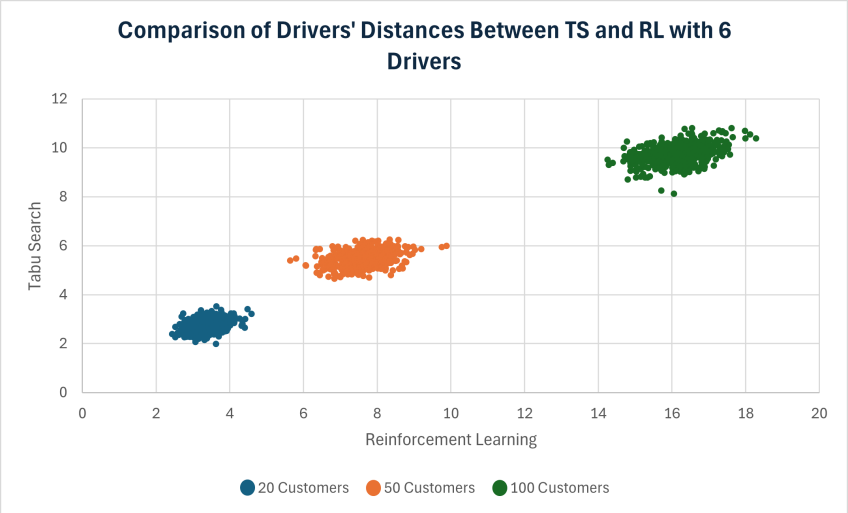
It is evident that using TS to solve the solution can reduce the total distance travelled by the vehicles across multiple customers.



(a) 4 Drivers



(b) 5 Drivers



(c) 6 Drivers

Figure 5.34 Scatter plot comparing Drivers' Distances Between TS and RL

A comparison of the average running times for both TS and RL is illustrated in Figures 5.35 to 5.37 for instances consisting of 20, 50 and 100 customers. The results show that TS was faster when solving smaller problems, as shown in instances with 20 customers, but struggled with larger problems, as shown in instances with 50 and 100 customers. As the number of customers increased, the TS computation time increased exponentially due to the increased complexity and size of the search space, requiring it to spend more time exploring and evaluating each candidate solution at each iteration.

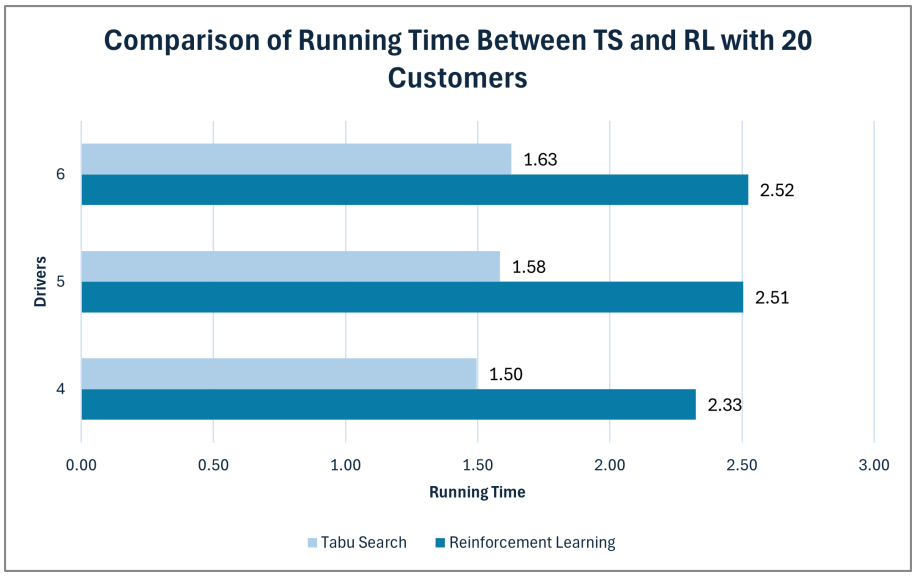


Figure 5.35 Comparing Average Running Time between TS and RL with 20 customers across each driver instance

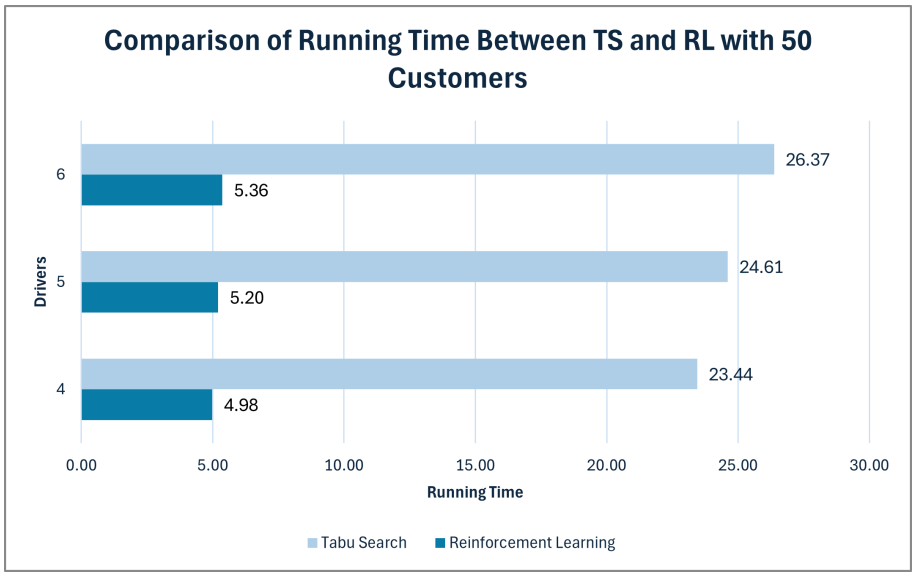


Figure 5.36 Comparing Average Running Time between TS and RL with 50 customers across each driver instance

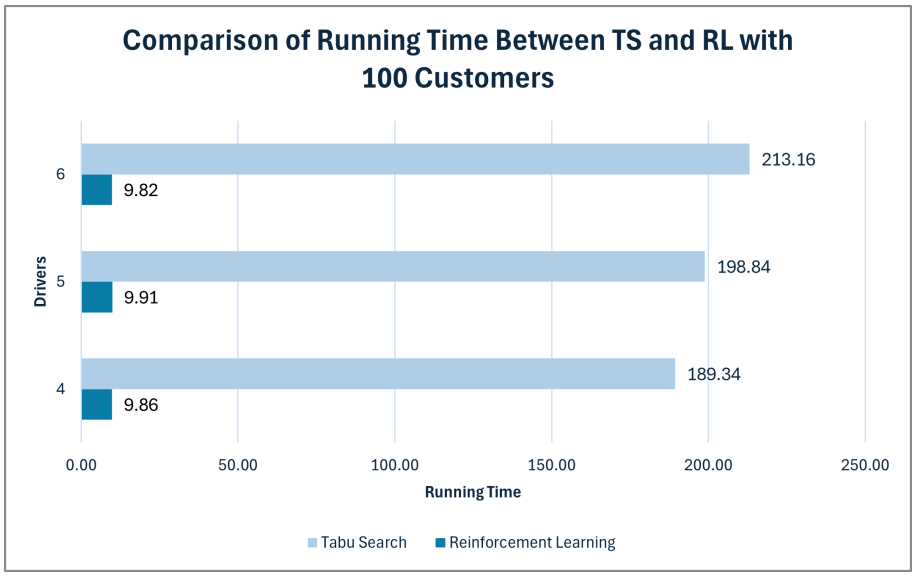


Figure 5.37 Comparing Average Running Time between TS and RL with 100 customers across each driver instance

Scatter plots were used to compare the running time performance of TS and RL when using the same dataset. Figures 5.38 to 5.40 illustrate the running time performance across instances with 4, 5 and 6 drivers, respectively. Each data point represents a sample from the dataset, indicating its performance when using RL (x-axis) and TS (y-axis).

The scatter plots show that across each driver instance, the points belonging to 20 customers form a nearly straight line with a narrow height across the x-axis. This indicates that in instances with 20 customers, the RL running time had a higher range of running time values than that of TS. When comparing the data points belonging to 50 customers, there is a cluster of data where the values on the y-axis are larger than those on the x-axis, suggesting that TS overall required more computational time than RL. The data points belonging to 100 customers for each driver instance are spread across the y-axis while remaining within a narrow range on the x-axis, showing a considerable variation in running time when using TS. The TS values on the y-axis were also considerably higher than those of RL on the x-axis, indicating a large increase in computation time compared to RL. This confirms that RL outperformed TS in terms of execution time on larger instances, highlighting RL's ability to handle large-scale MVRRPP instances by learning optimal policies that generalise different problem sizes.

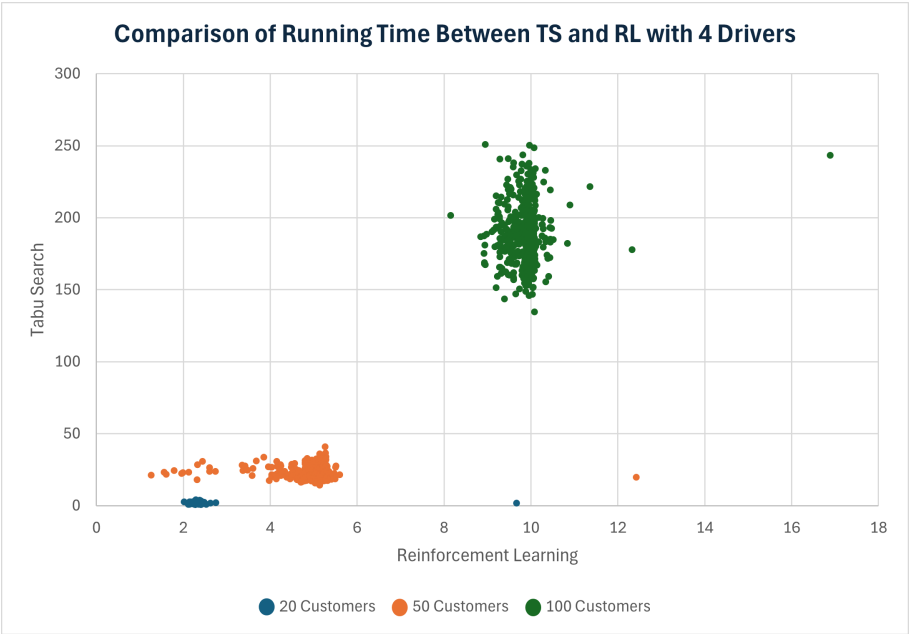


Figure 5.38 Scatter Plot comparing the Running Time between TS and RL with 4 drivers across multiple customers

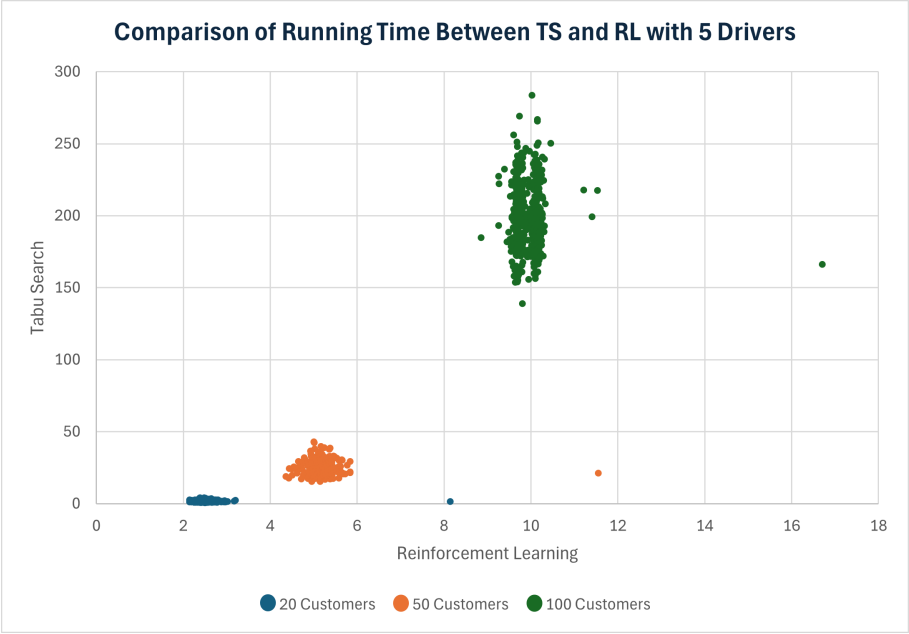


Figure 5.39 Scatter Plot comparing the Running Time between TS and RL with 5 drivers across multiple customers

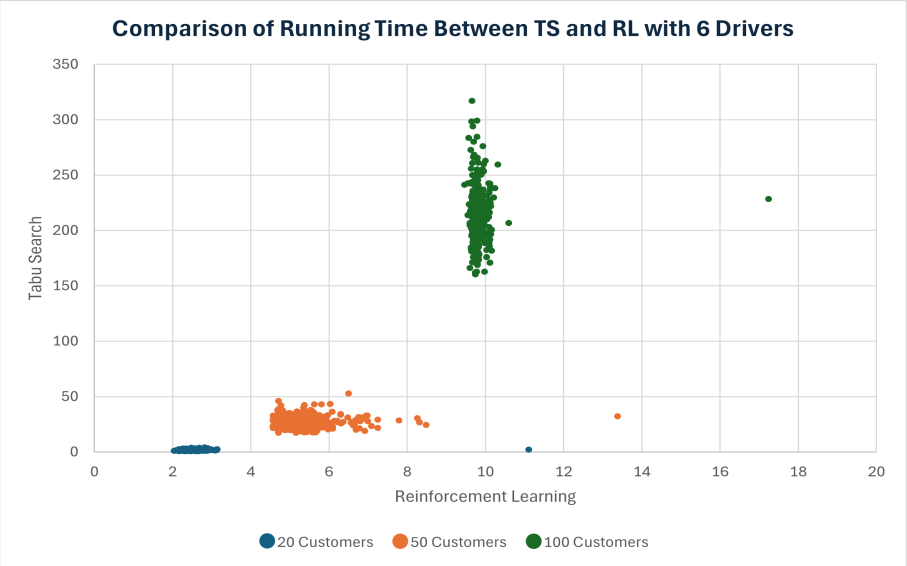


Figure 5.40 Scatter Plot comparing the Running Time between TS and RL with 6 drivers across multiple customers

5.3 Objective 3 - Initialising Tabu Search with a solution generated using Reinforcement Learning

The third experiment involved using the solution obtained from RL in Experiment 2 as the initial solution for TS. The performance was then evaluated and compared to the previous objectives using the respective datasets of each scenario.

Figure 5.41 illustrates a graph instance of 20 customers and 4 drivers while Figure 5.42 illustrates the final solution achieved when using TS with RL on this instance. A graph with each driver's route from the final solution is depicted in Figure 5.43.

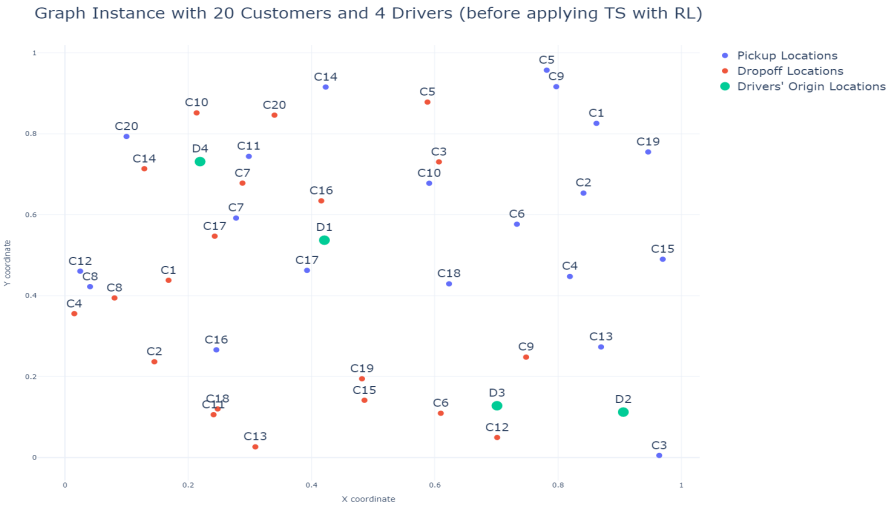


Figure 5.41 Graph Instance with 20 Customers and 4 Drivers before using TS with RL

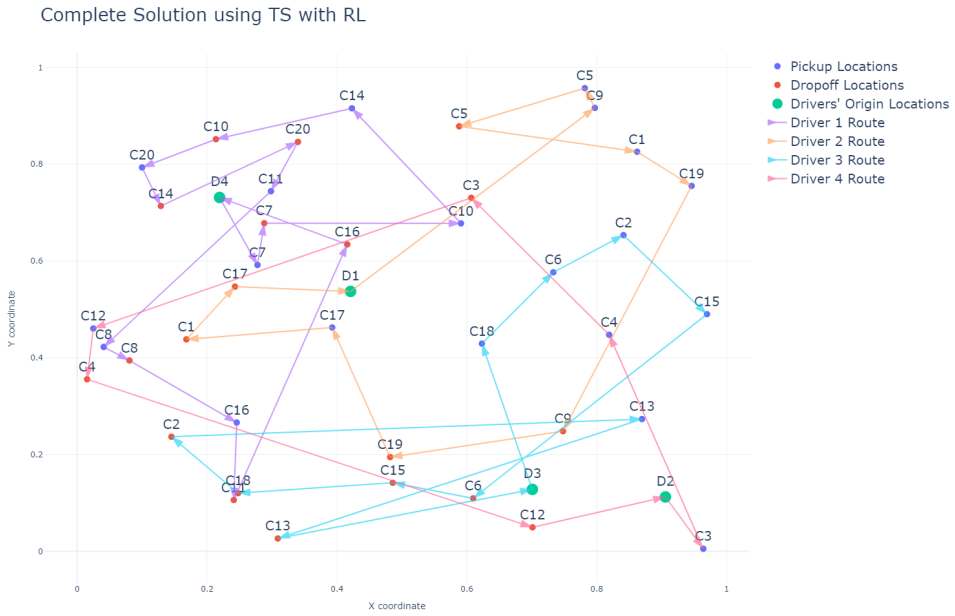


Figure 5.42 Final Solution after applying TS with RL on the graph instance of 20 customers and 4 drivers

(a) Driver 1 Route

(b) Driver 2 Route

(c) Driver 3 Route

(d) Driver 4 Route

Figure 5.43 Drivers' Routes obtained using TS with RL on an instance with 4 drivers and 20 customers

Figure 5.44 illustrates the waiting time performance when using TS with RL with instances of different drivers and customers. The box plot shows that for each driver instance, the waiting time increased as more customers were added. The box plots also showed that the overall waiting time decreased as more drivers were added to an instance with the same number of customers.

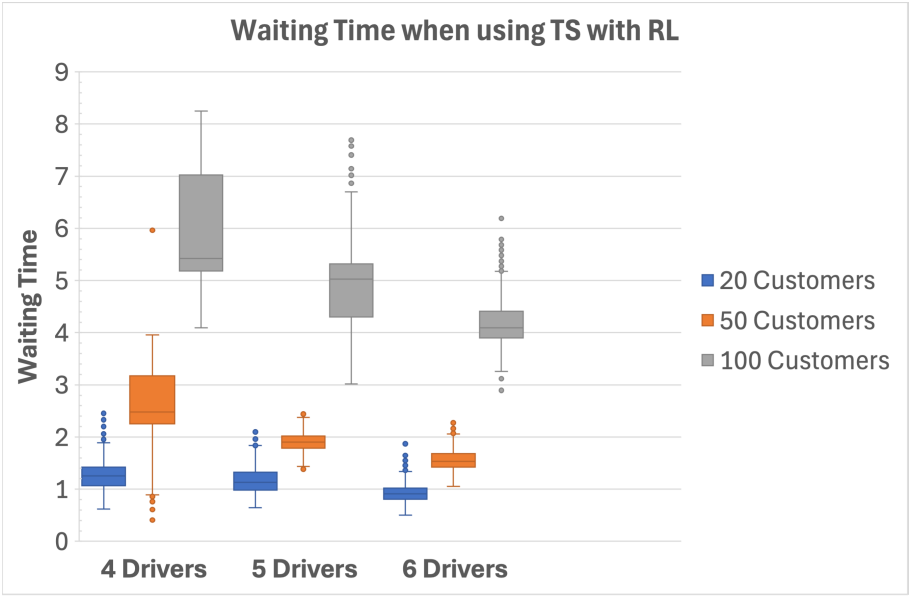


Figure 5.44 Box Plot with the Waiting Time across different instances of drivers and customers using TS with RL

As illustrated in Figure 5.45, the travel impact time increased with the addition of customers, as the drivers were required to make more stops. In instances with 20 and 50 customers, the travel impact time decreased as more drivers were available, indicating that the customers were being distributed along the routes. However, in instances with 100 customers, the travel impact time remained consistent and varied slightly across instances with 4, 5 and 6 drivers.

The box plot in Figure 5.46 illustrates the distance travelled across different instances. The box plot shows that the distance travelled by each group of drivers increased when more customers were added. This was expected since the driver needed to make additional stops for each customer's pickup and dropoff location, increasing the overall distance travelled. As evident in the box plot, the increase in drivers helped distribute customers along different routes.

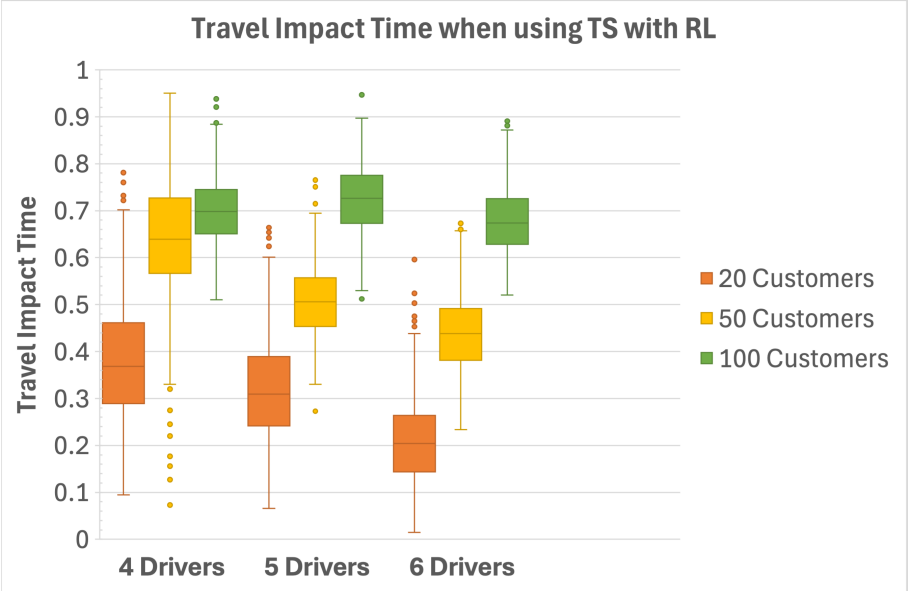


Figure 5.45 Box Plot with the Travel Impact Time across different instances of drivers and customers using TS with RL

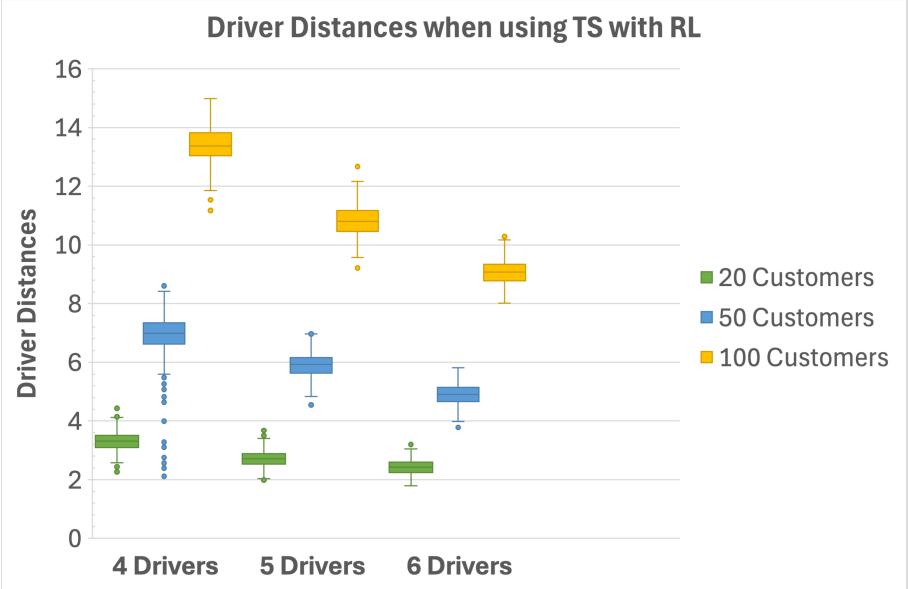


Figure 5.46 Box Plot with the Drivers' Distances across different instances of drivers and customers using TS with RL

Figure 5.47 shows the number of iterations taken by the TS with RL model to find the optimal solution. More iterations were required to find the optimal solution in instances with larger customers, for each driver group. However, the number of iterations remained consistent across each driver group with the same number of customers.

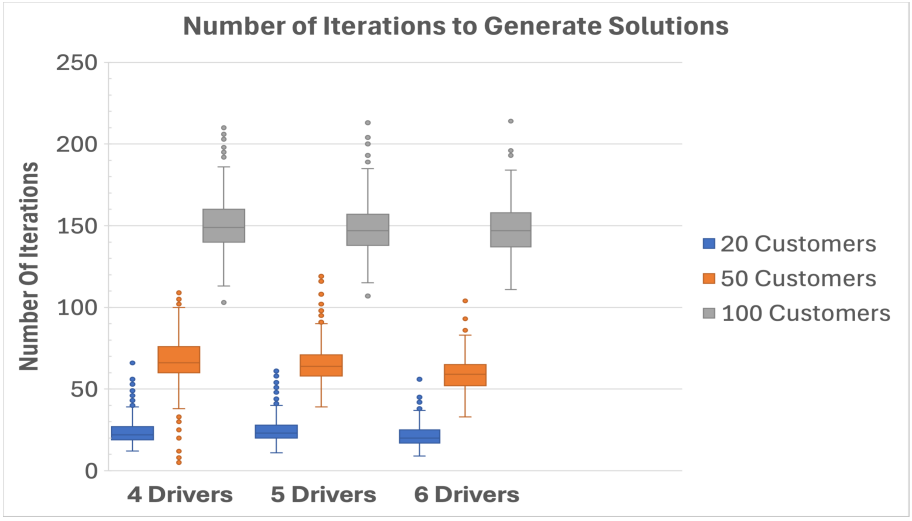
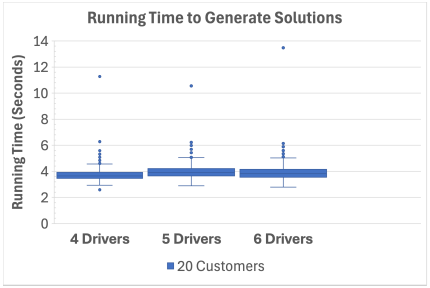
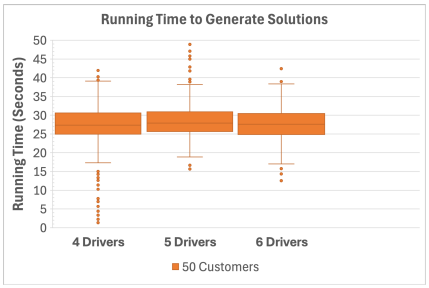


Figure 5.47 Box Plot with the number of iterations when using TS with RL

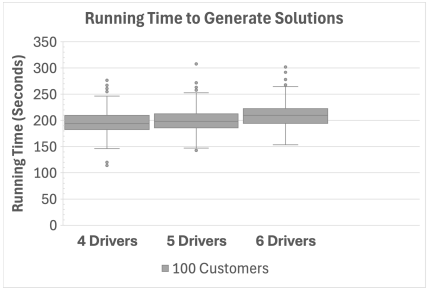
Figure 5.48 shows the running time taken by the TS with RL to generate the best solutions. The box plots show that the running time experienced exponential growth when adding customers, suggesting that the TS with RL model required more computation time when finding the optimal solution in the larger search space.



(a) 20 Customers



(b) 50 Customers



(c) 100 Customers

Figure 5.48 Running Time to generate solutions using TS

Figure 5.49 to 5.51 illustrate bar charts representing the average waiting time performance of TS using RL compared with the waiting time of TS (Objective 1) and RL (Objective 2) across 20, 50 and 100 customers, respectively. The lower the bar, the better the performance since the aim is to minimise each metric. The bar charts show a similar waiting time performance between TS and TS with RL. Instances with customers of 20 (Figure 5.49) and 100 (Figure 5.51) indicate that TS performed best in terms of waiting time, followed by TS with RL and finally RL. In instances with 50 customers (Figure 5.50), the TS with RL outperformed TS while RL followed last.

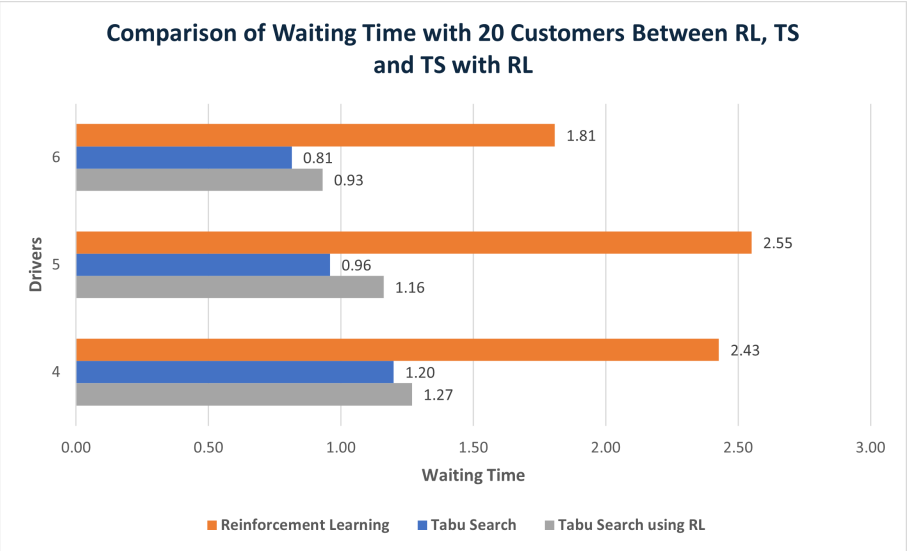


Figure 5.49 Comparing Average Waiting Time with 20 customers across each driver instance between RL, TS and TS with RL

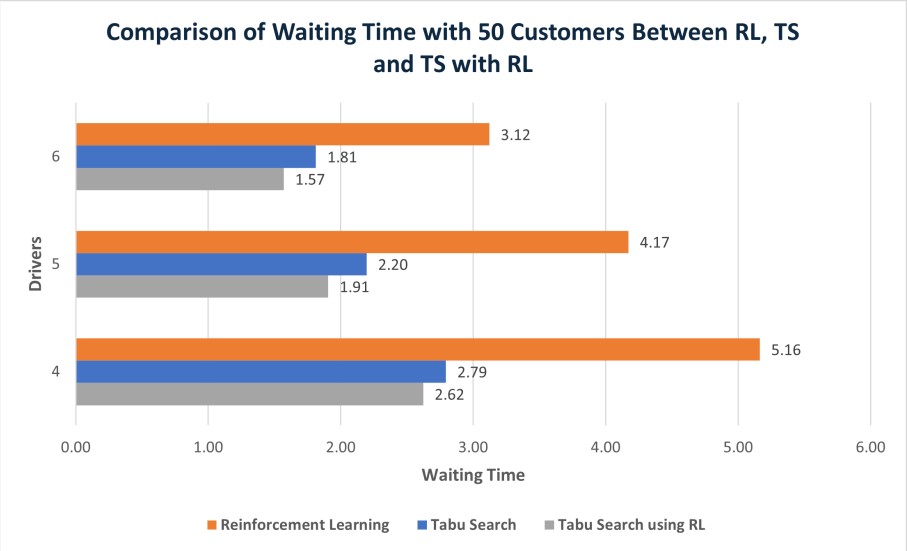


Figure 5.50 Comparing Average Waiting Time with 50 customers across each driver instance between RL, TS and TS with RL

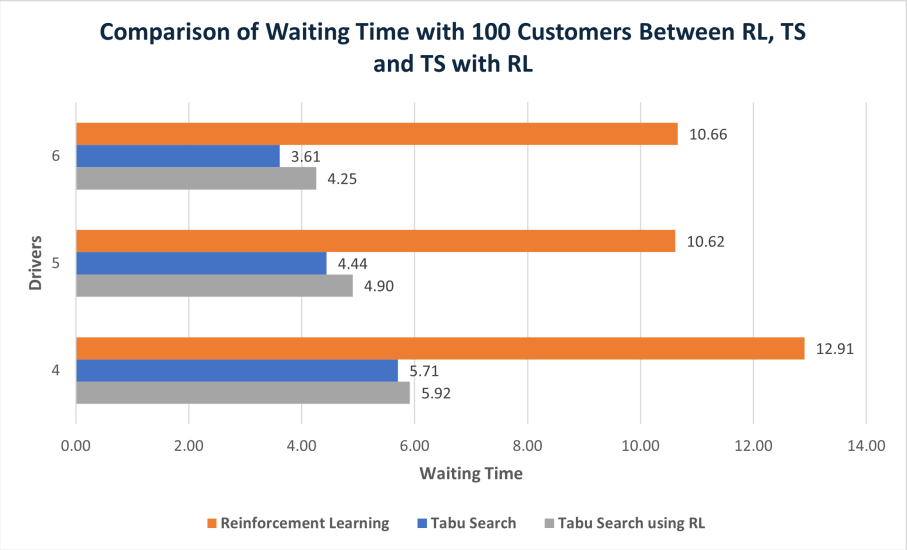
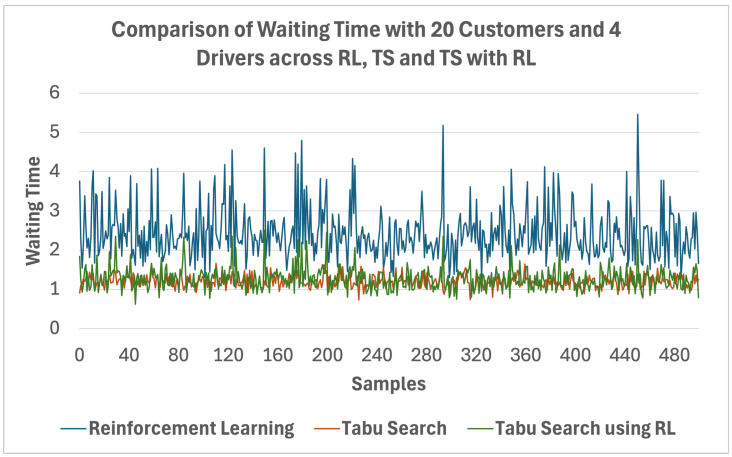


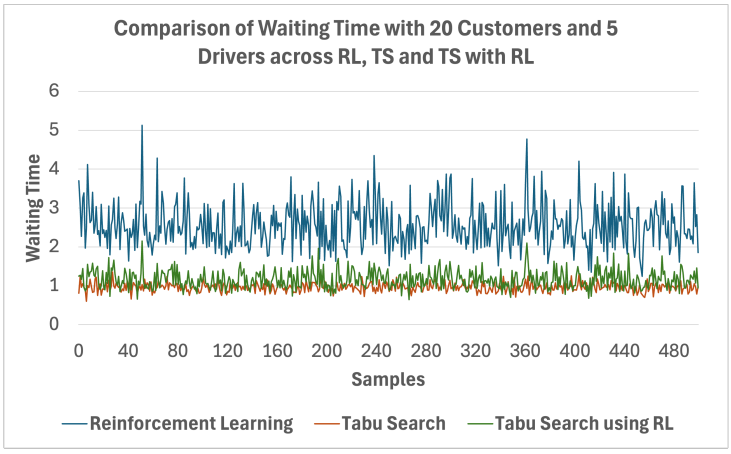
Figure 5.51 Comparing Average Waiting Time with 100 customers across each driver instance between RL, TS and TS with RL

Figures 5.52 to 5.54 are line charts representing the waiting times across all experiments for each sample using the same dataset. Each line chart represents the waiting time with 20, 50, and 100 customers across 4, 5 and 6 drivers. All line charts show fluctuations in waiting time across different models, highlighting how each problem varies in complexity with each sample.

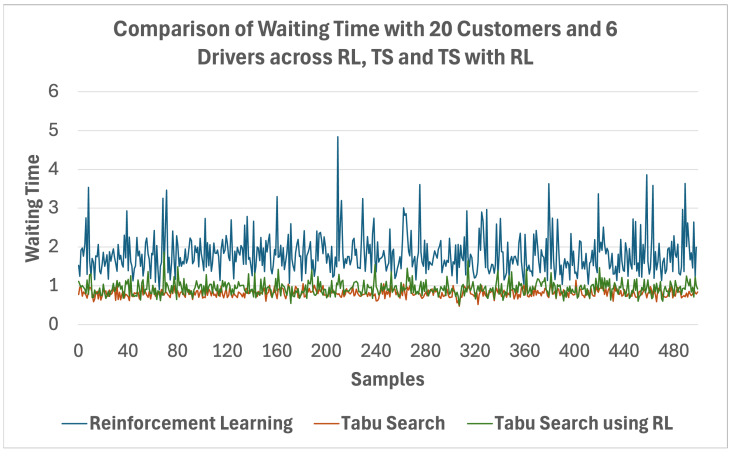
Figure 5.52 shows the waiting time for each model across 20 customers. The line charts shows that the performance of TS and TS with RL across each driver instance was relatively similar, with TS with RL showing a slightly higher range in values. Figure 5.53 shows the waiting time across 50 customers for each model. These line charts show that TS with RL had the lowest overall range in average waiting time values when compared to TS and RL. The line representing RL showed an overall decrease in values between 4 and 5 drivers, indicating a performance improvement, with 6 drivers also remaining similar in terms of waiting time performance to that of 5 drivers. Figure 5.54 shows the waiting time performance across 100 customers. In these line charts, TS had the lowest range in waiting time values, with TS with RL having slightly higher values. RL had the highest waiting time values, which however decreased as the number of drivers increased.



(a) 4 Drivers

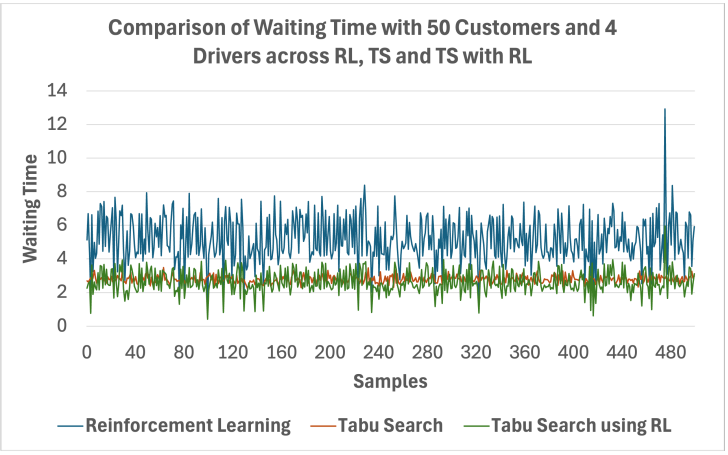


(b) 5 Drivers

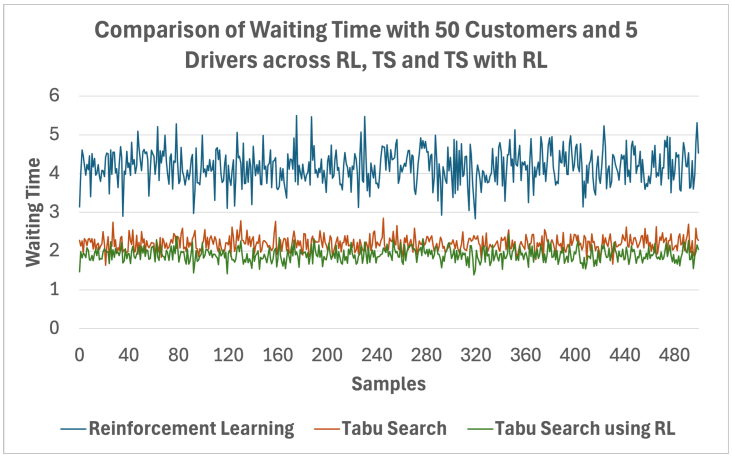


(c) 6 Drivers

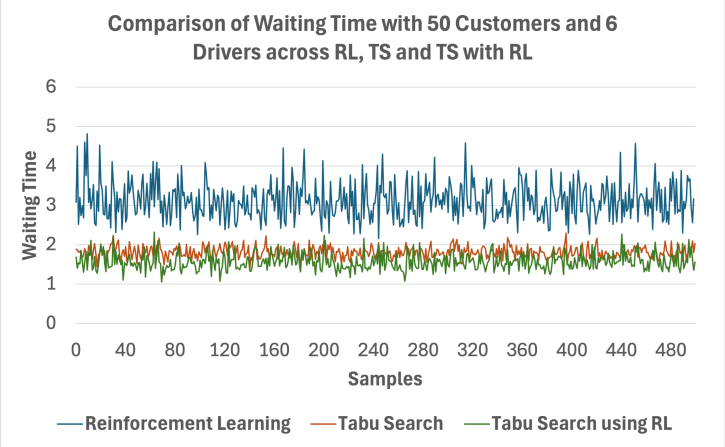
Figure 5.52 Line Chart comparing Waiting Time between RL, TS and TS with RL across samples with 20 customers



(a) 4 Drivers

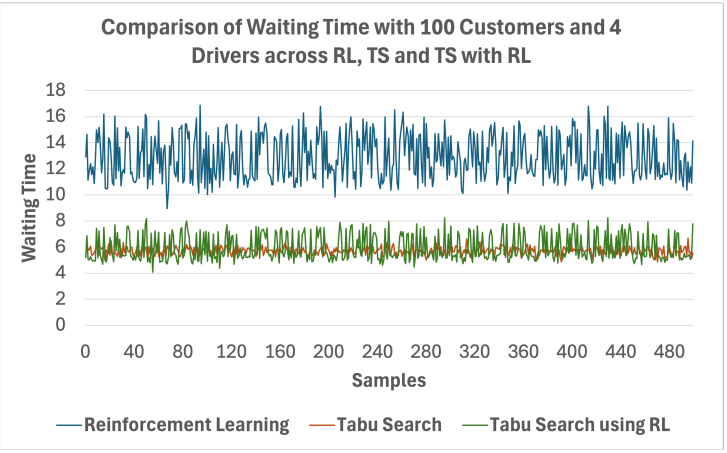


(b) 5 Drivers

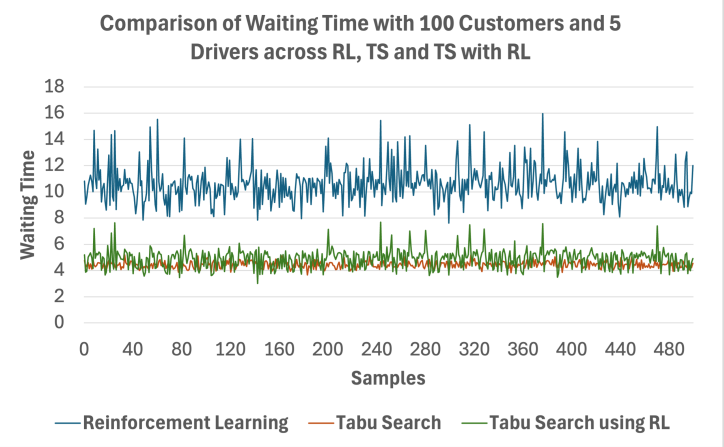


(c) 6 Drivers

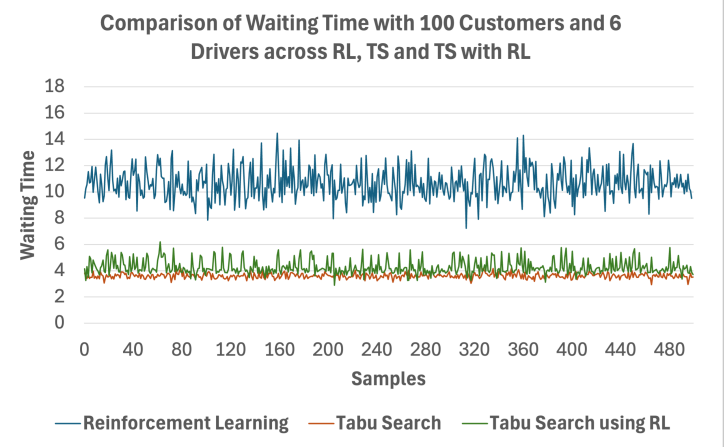
Figure 5.53 Line Chart comparing Waiting Time between RL, TS and TS with RL across samples with 50 customers



(a) 4 Drivers



(b) 5 Drivers



(c) 6 Drivers

Figure 5.54 Line Chart comparing Waiting Time between RL, TS and TS with RL across samples with 100 customers

Figures 5.55 to 5.57 display the average travel impact time for the three experiments when evaluated on 20, 50 and 100 customers across different drivers. As observed in Figure 5.55, TS achieved the lowest travel impact time average, followed by TS with RL, and RL with the highest average. On the other hand, Figure 5.56 and Figure 5.57 reveal a similar performance between TS and TS with RL, with, at times, TS with RL surpassing TS. Overall, the average travel impact time increased with the addition of customers across all experiments. In all figures examined, RL consistently exhibited the highest time customers spent from pickup to dropoff compared to the TS models. This observation highlights the challenges faced by RL in optimisation vehicle routing solutions effectively.

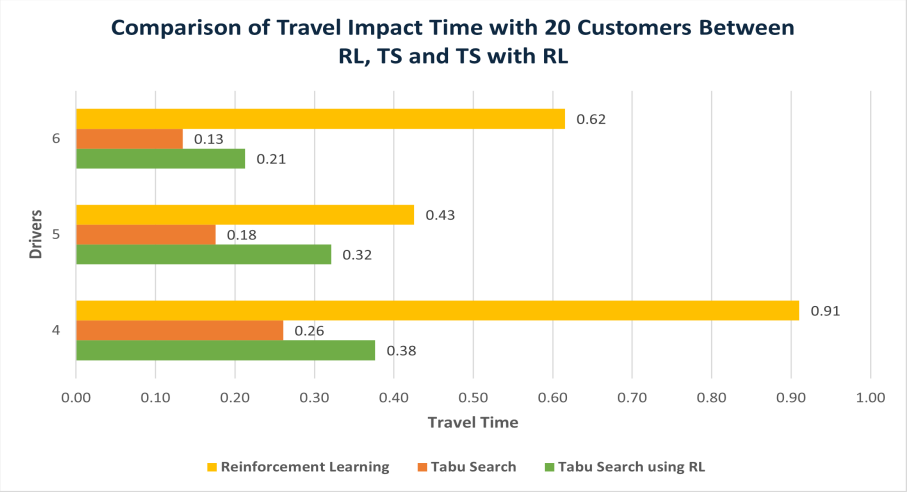


Figure 5.55 Comparing Average Travel Impact Time with 20 customers across each driver instance between RL, TS and TS with RL

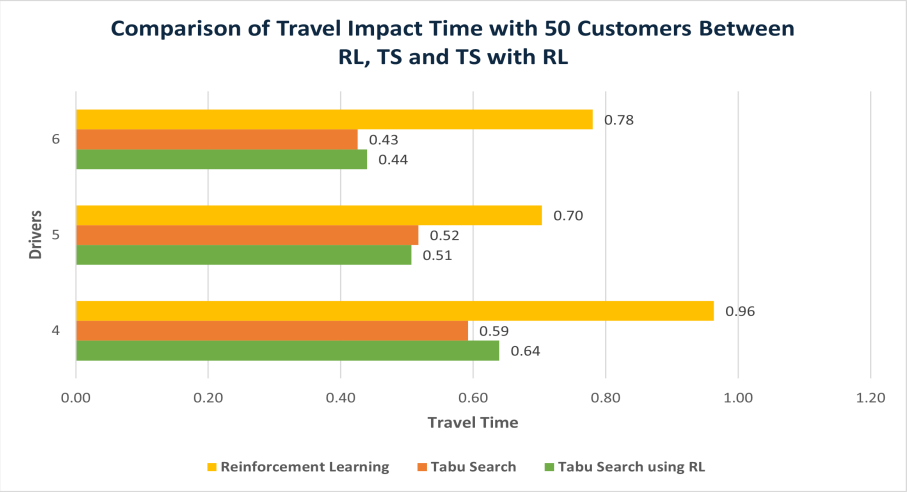


Figure 5.56 Comparing Average Travel Impact Time with 50 customers across each driver instance between RL, TS and TS with RL

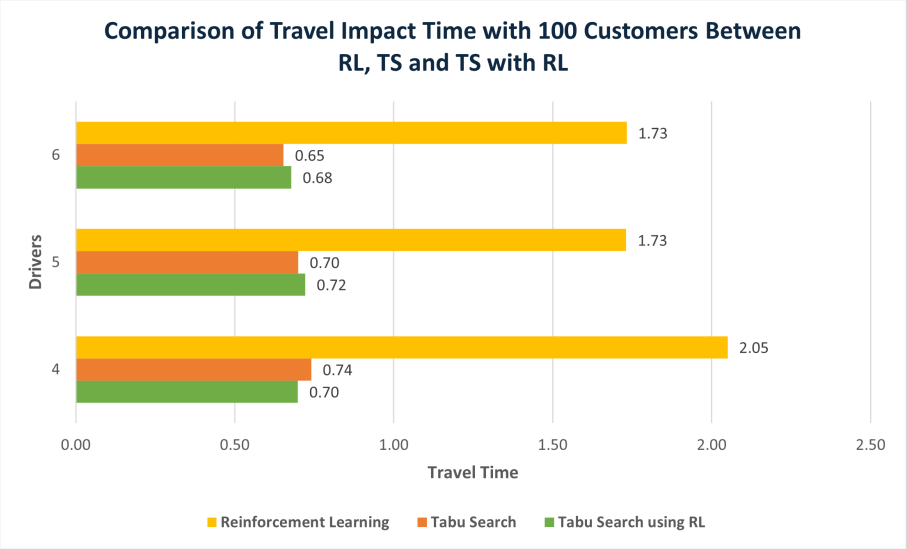
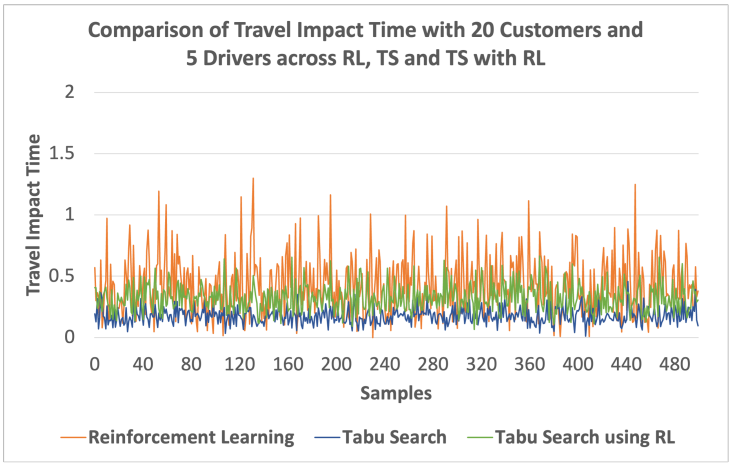


Figure 5.57 Comparing Average Travel Impact Time with 100 customers across each driver instance between RL, TS and TS with RL

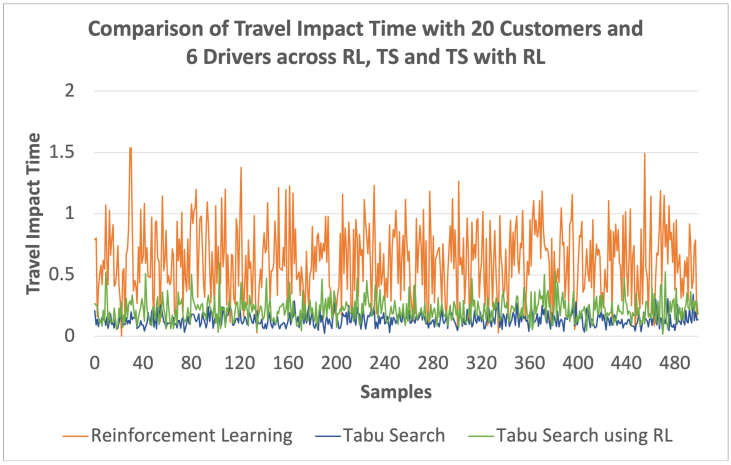
Figures 5.58 to 5.60 illustrate line charts that depict the travel impact time performance for each sample in the dataset for the three experiments. Across all figures, the line depicting RL's performance showed large spikes throughout, indicating substantial variations in the travel impact time across different instances. The variations suggest that RL faced challenges in minimising customers' time between pickup and dropoff while optimising routing efficiency.

In Figure 5.58, TS showed the best performance, achieving the lowest range and travel impact time values across each group of drivers with 20 customers, whilst TS with RL had slightly higher values and range followed by RL. Figure 5.59 and Figure 5.60 showed similar performance between TS and TS with RL across all groups, excluding the instance with 100 customers and 6 drivers, where the performance was slightly better when using TS with a lower range in values.

(a) 4 Drivers

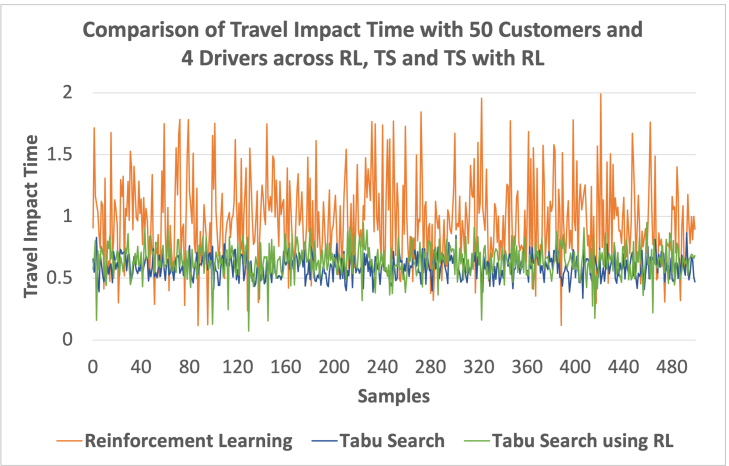


(b) 5 Drivers

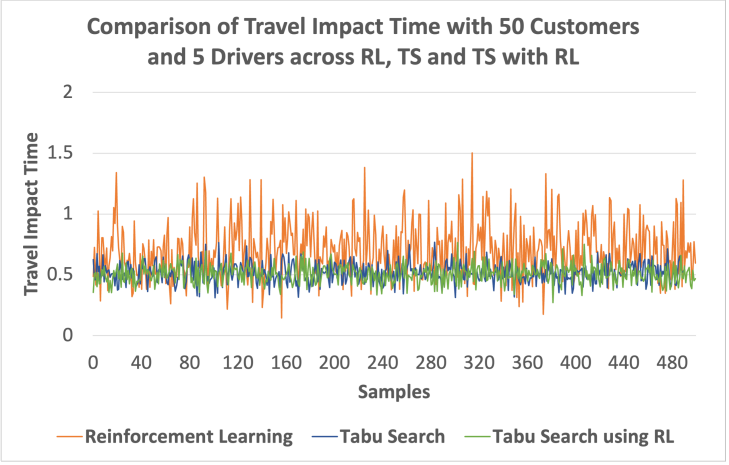


(c) 6 Drivers

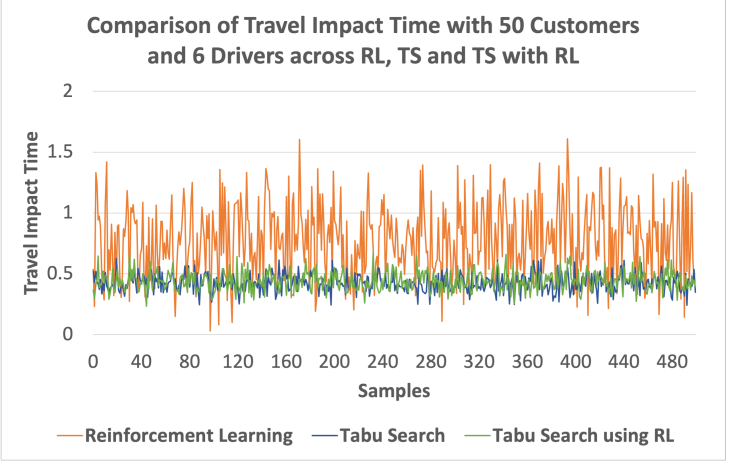
Figure 5.58 Line Chart comparing Travel Impact Time between RL, TS and TS with RL across samples with 20 customers



(a) 4 Drivers

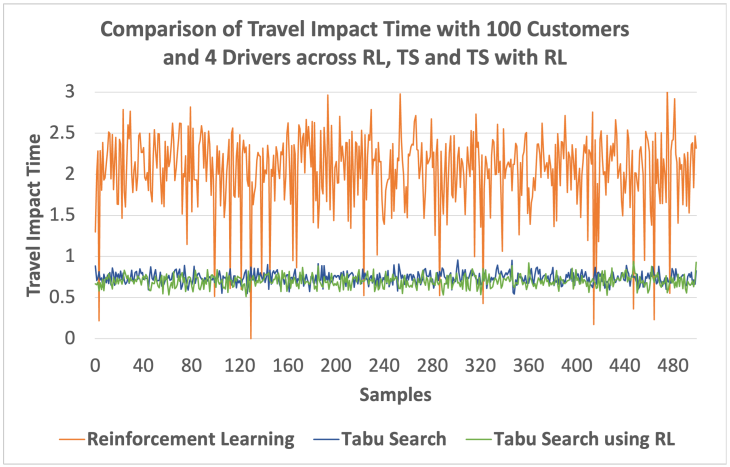


(b) 5 Drivers

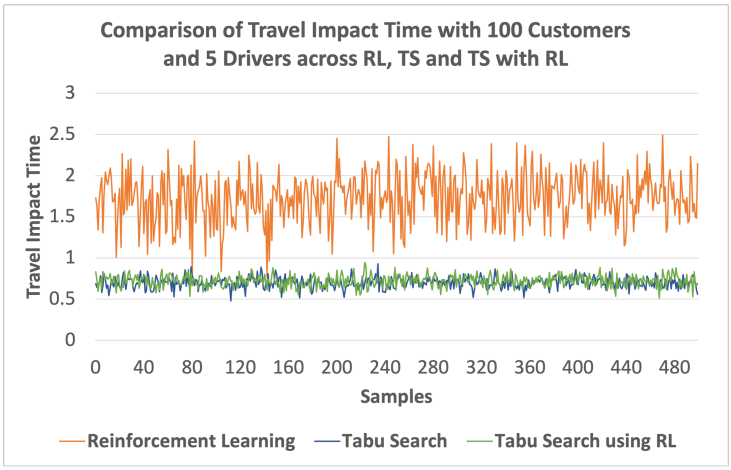


(c) 6 Drivers

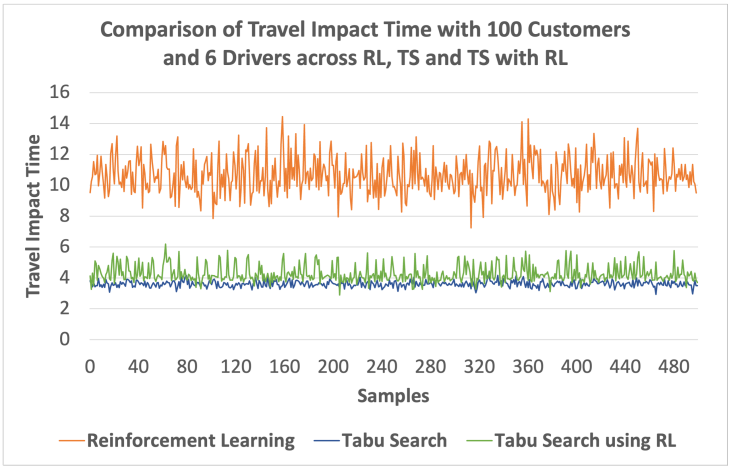
Figure 5.59 Line Chart comparing Travel Impact Time between RL, TS and TS with RL across samples with 50 customers



(a) 4 Drivers



(b) 5 Drivers



(c) 6 Drivers

Figure 5.60 Line Chart comparing Travel Impact Time between RL, TS and TS with RL across samples with 100 customers

The bar charts in Figures 5.61 to 5.63 depict the total average distance travelled when evaluating the three experiments on groups of 20, 50 and 100 customers with different drivers. The bar charts indicate that TS with RL performed best with the least distance travelled across each instance with different numbers of customers and driver groups. The total distance travelled was slightly higher when using TS but still considerably lower than RL's. As expected, the bar charts indicate that as the number of drivers increases, the distance travelled decreases across all models with the same number of customers. Nevertheless, the total average distance travelled increases across all models as the number of customers increases.

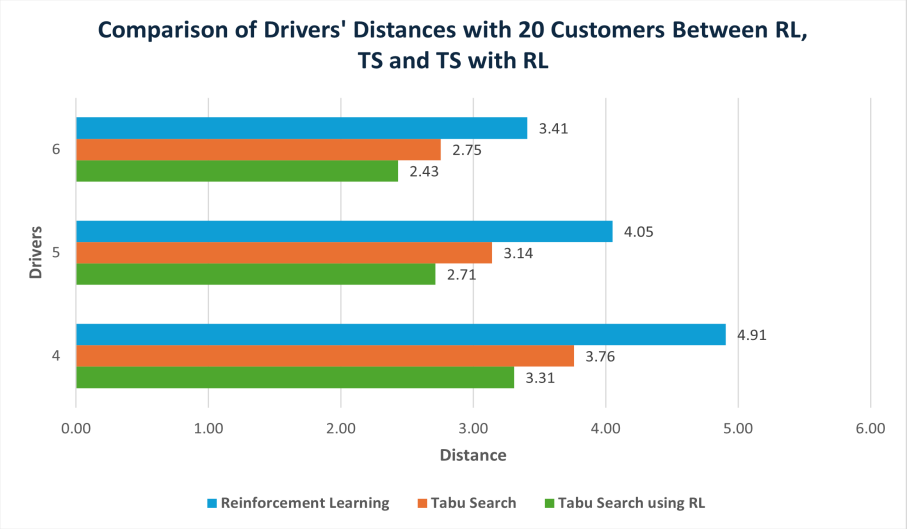


Figure 5.61 Comparing Average Drivers' Distances with 20 customers across each driver instance between RL, TS and TS with RL

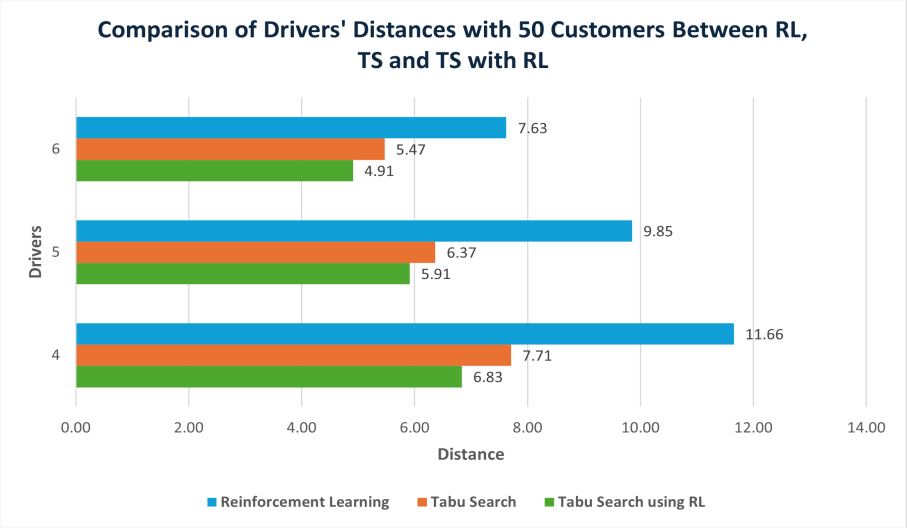


Figure 5.62 Comparing Average Drivers' Distances with 50 customers across each driver instance between RL, TS and TS with RL

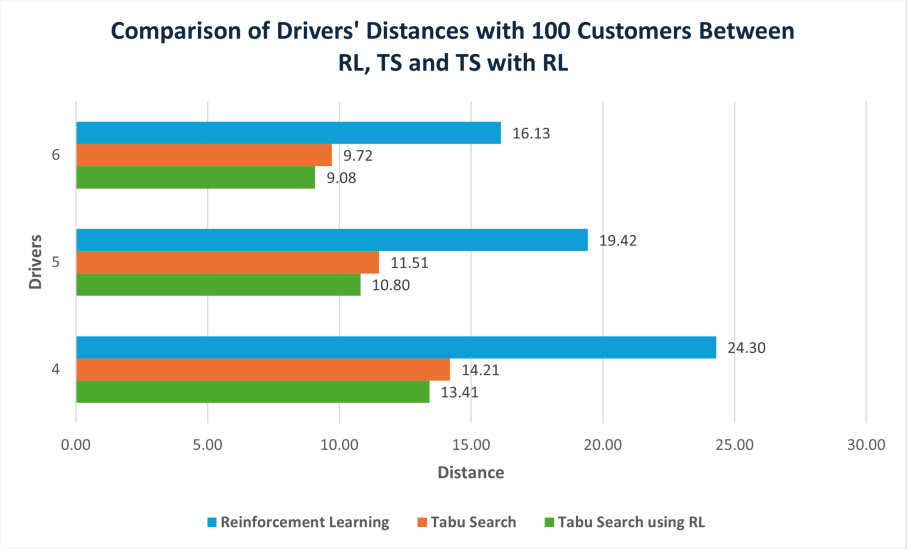
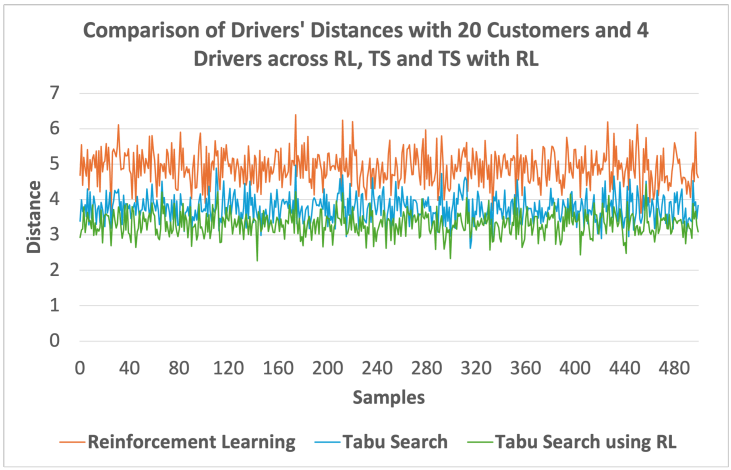
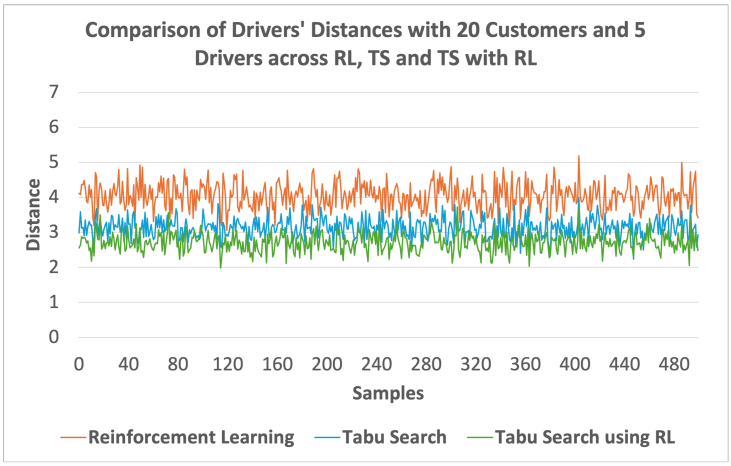


Figure 5.63 Comparing Average Drivers' Distances with 100 customers across each driver instance between RL, TS and TS with RL

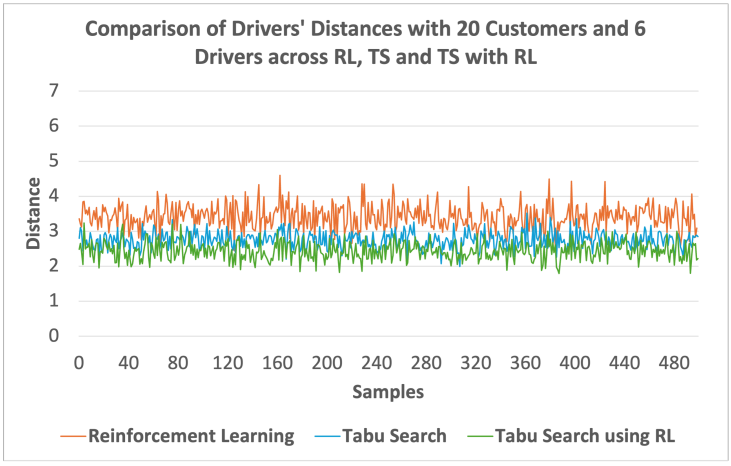
The line charts in Figures 5.64 to 5.66 depict the total distance travelled achieved for each sample in the dataset across the three experiments. The line charts indicate that the line depicting the performance of TS with RL had the lowest set of values across each group of drivers and customers, followed by TS, which had slightly higher distance values, and RL, which had the highest values. The line charts with 50 customers and 4 drivers showed high variations in the total distance travelled when using RL and TS with RL. This suggests that the solution obtained using RL may not have been optimal in terms of distance, hindering the TS with RL's ability to find a better solution, leading to spikes in the distance metric as it attempts to navigate and refine the suboptimal solution. The line charts also show a wider gap between RL's performance and other models as customer numbers increased. This suggests that the RL model may have faced challenges in scaling to larger problem instances with the increase in customers.



(a) 4 Drivers

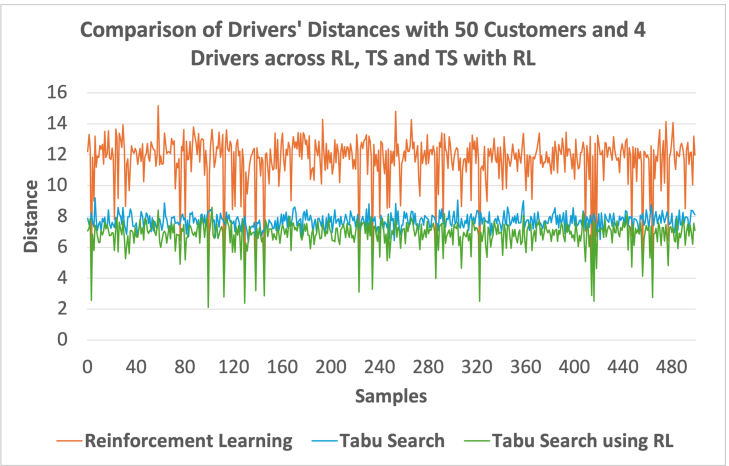


(b) 5 Drivers

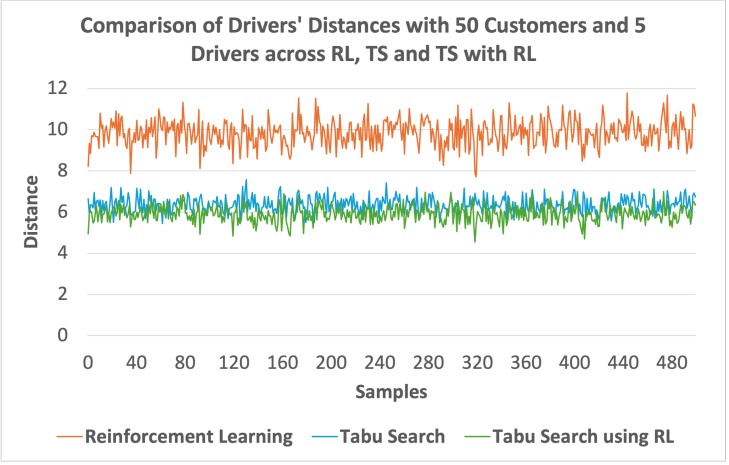


(c) 6 Drivers

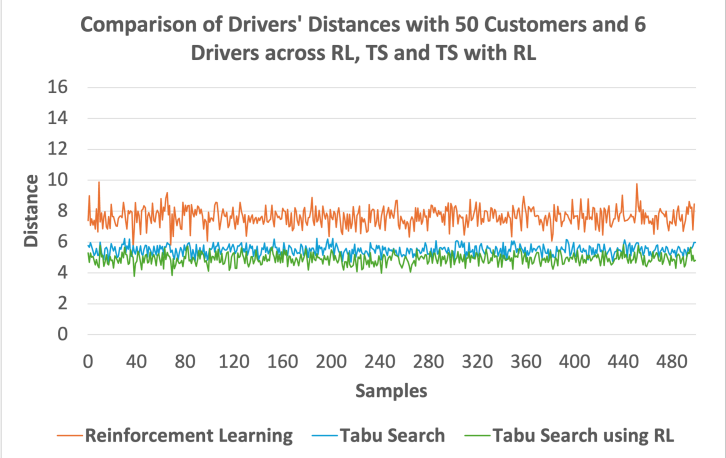
Figure 5.64 Line Chart comparing Drivers' Distances between RL, TS and TS with RL across samples with 20 customers



(a) 4 Drivers

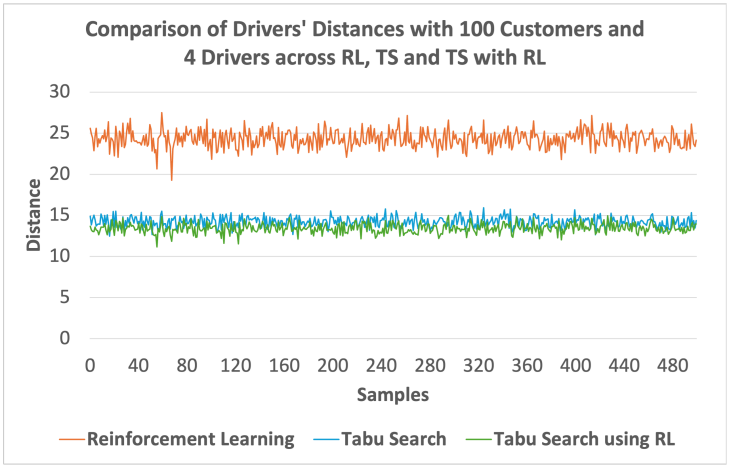


(b) 5 Drivers

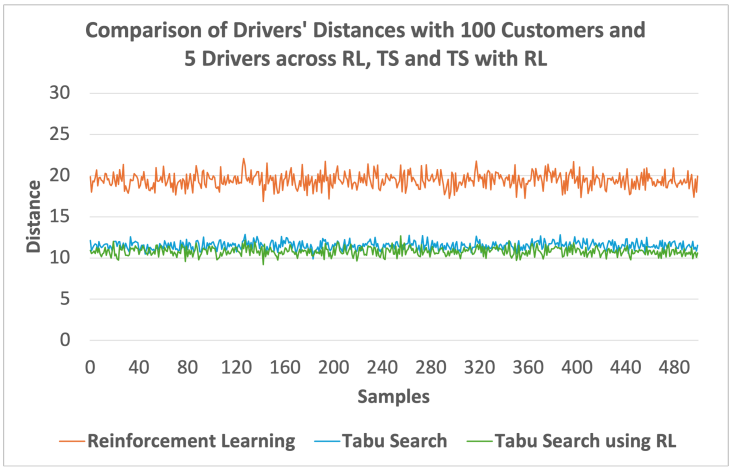


(c) 6 Drivers

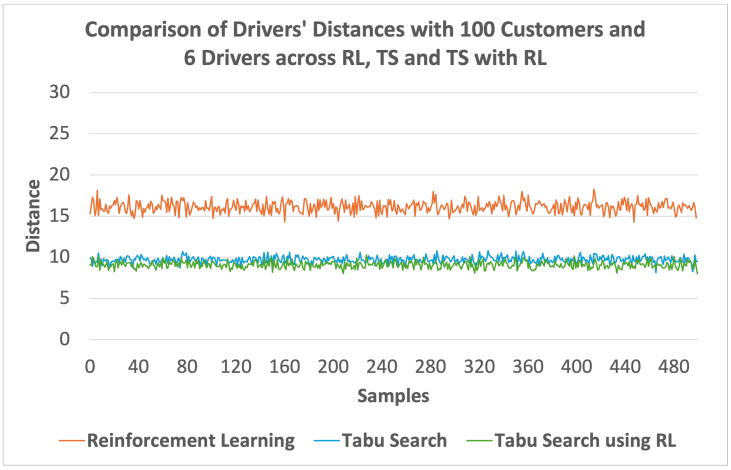
Figure 5.65 Line Chart comparing Drivers' Distances between RL, TS and TS with RL across samples with 50 customers



(a) 4 Drivers



(b) 5 Drivers



(c) 6 Drivers

Figure 5.66 Line Chart comparing Drivers' Distances between RL, TS and TS with RL across samples with 100 customers

Figures 5.67 to 5.69 show the average running time performance across all three models with groups of 20, 50 and 100 customers, respectively. The TS with RL is presented as a stacked bar to show the time taken to generate the initial solution using RL, followed by the time taken to use that initial solution to generate the final solution using TS.

Figure 5.67 shows that in instances with 20 customers, TS was the fastest and required the least computation time out of the three models, followed by RL and then TS with RL. The stacked bar belonging to TS with RL indicated that a large portion of its running time was due to RL taking a significant amount of time to generate the initial solution. However, in instances with 50 and 100 customers, as shown in Figure 5.68 and Figure 5.69, respectively, RL significantly outperformed the other models in terms of computational time, demonstrating its ability to handle larger problems more effectively by adapting to complex and dynamic environments. Both TS models performed similarly in terms of running time in these instances. They both exhibited a trend of increasing computation time with the addition of customers, indicating potential limitations in efficiently handling larger MVRRPP instances.

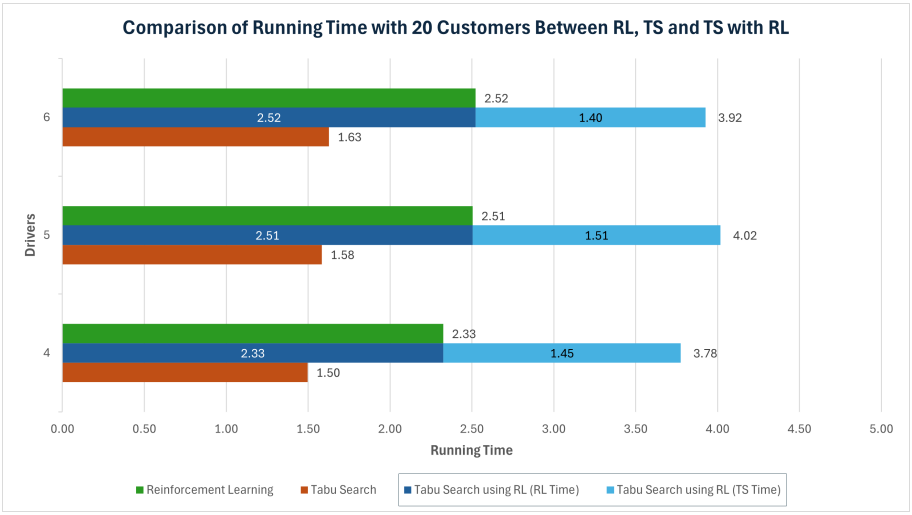


Figure 5.67 Comparing Average Running Time with 20 customers across each driver instance between RL, TS and TS with RL

The scatter plots in Figure 5.70 to 5.72 illustrate the performance of TS and TS-with-RL based on the time taken versus the number of iterations required to complete each sample in the dataset. The scatter plots show that the data points of each model formed a diagonal line, where the running time steadily increased as the number of iterations increased, indicating a consistent relationship between the two variables.

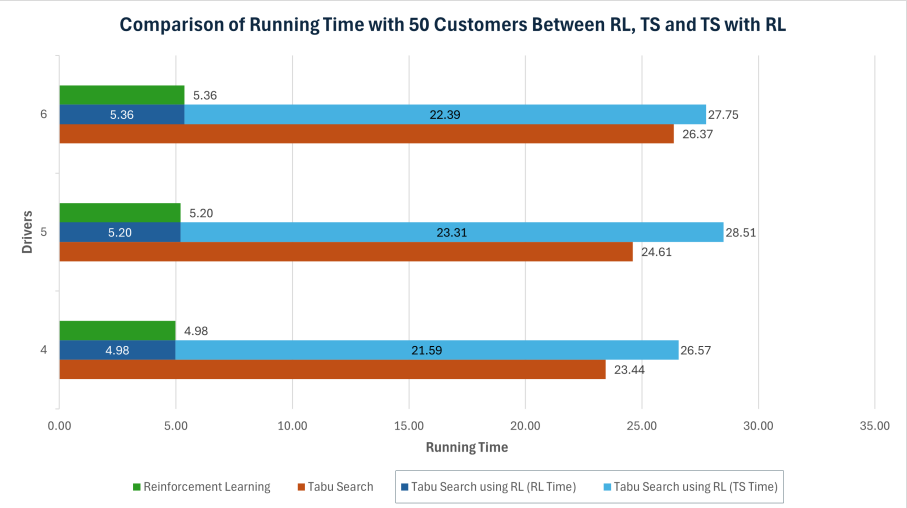


Figure 5.68 Comparing Average Running Time with 50 customers across each driver instance between RL, TS and TS with RL

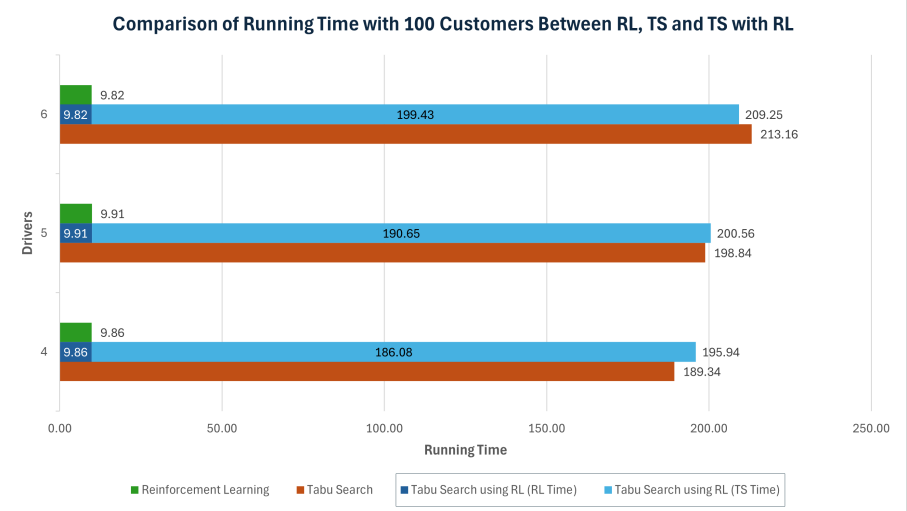
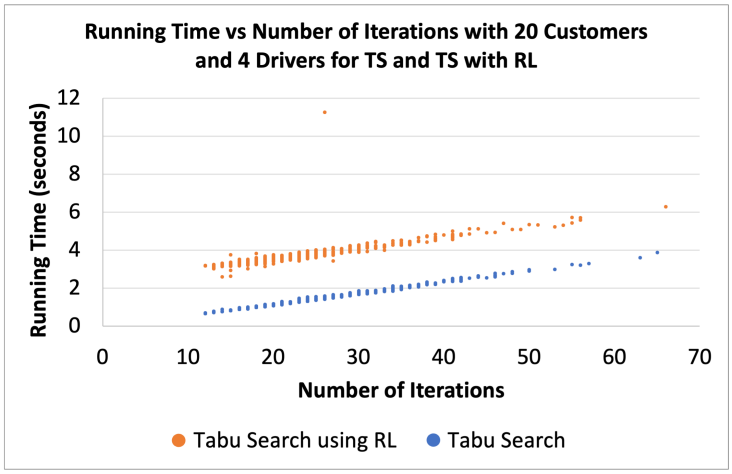
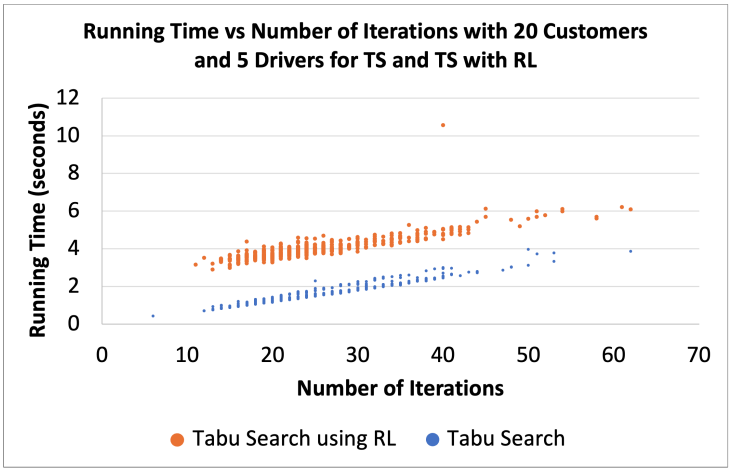


Figure 5.69 Comparing Average Running Time with 100 customers across each driver instance between RL, TS and TS with RL

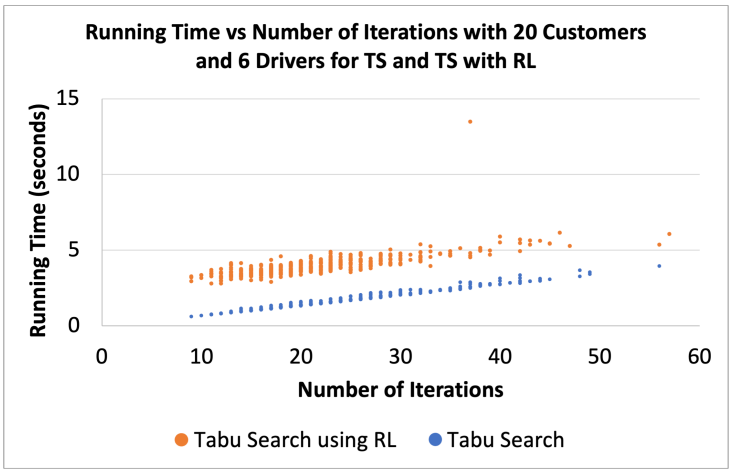
Lower diagonal lines belonging to those of TS, in instances of 20 customers and 50 customers, as shown in Figure 5.70 and Figure 5.71, respectively, indicate that the model completed the solution in less time using the same number of iterations, showing its efficiency over the TS with RL model. The scatter plots show that as the number of customers increases, the distance between the diagonal lines of each model decreases, suggesting a similar performance between models. This becomes apparent in instances with 100 customers, Figure 5.72, where data points of the two models overlap. Figure 5.72c shows that the data points belonging to TS with RL had lower overall values than TS, indicating that it was faster in finding the final solution.



(a) 4 Drivers

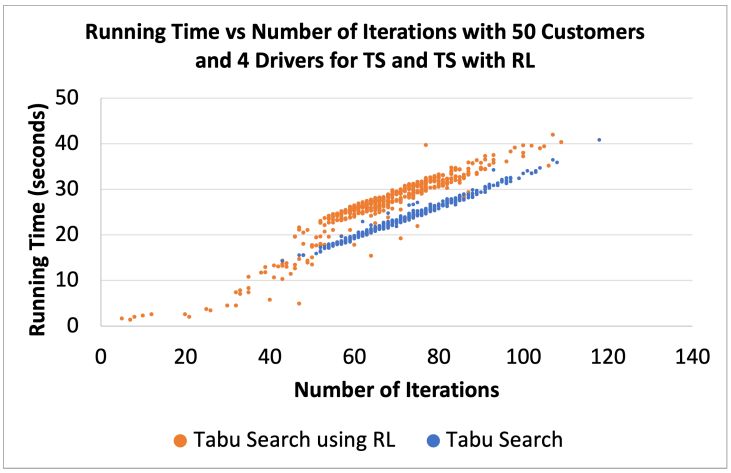


(b) 5 Drivers

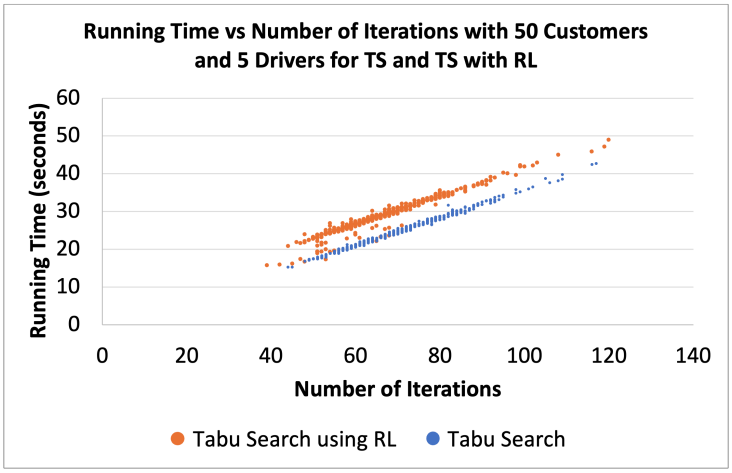


(c) 6 Drivers

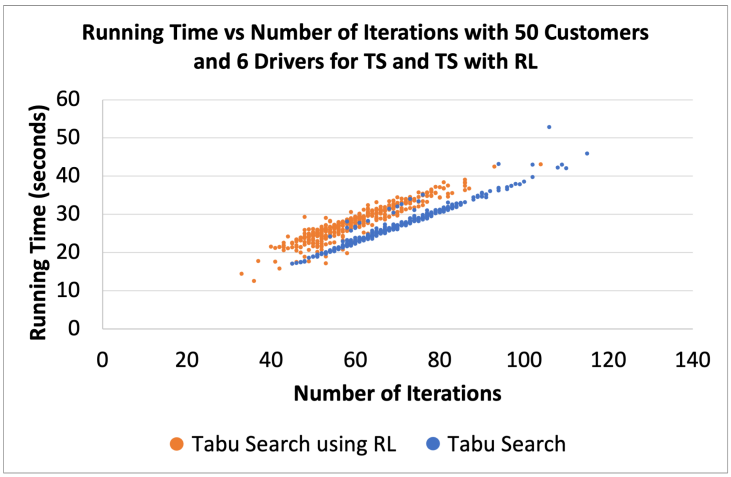
Figure 5.70 Scatter Plot comparing Running Time between RL, TS and TS with RL across samples with 20 customers



(a) 4 Drivers

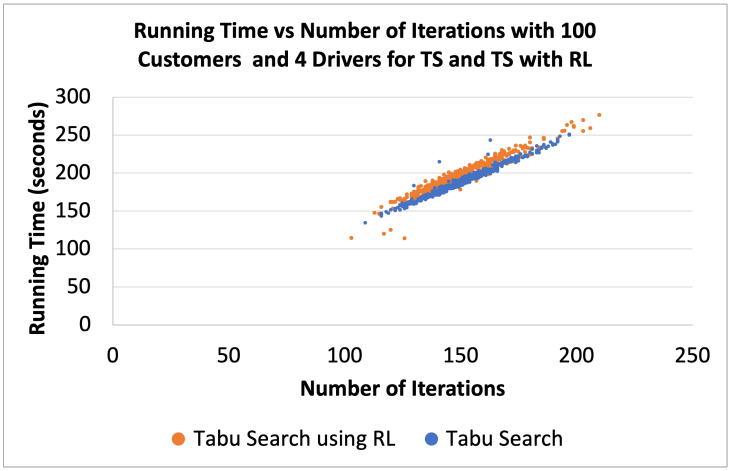


(b) 5 Drivers

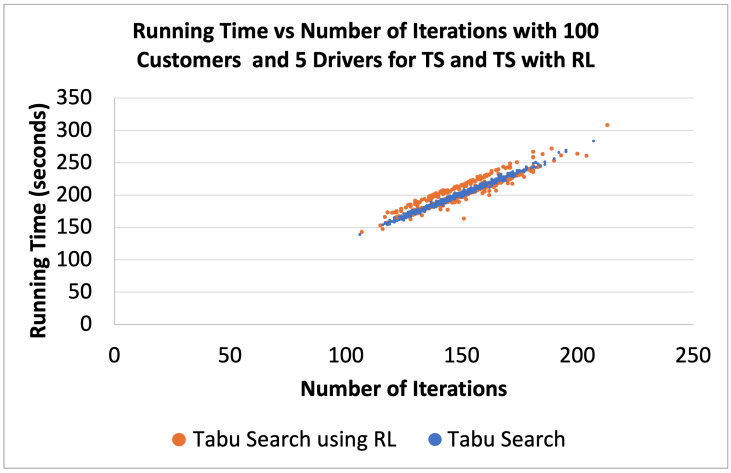


(c) 6 Drivers

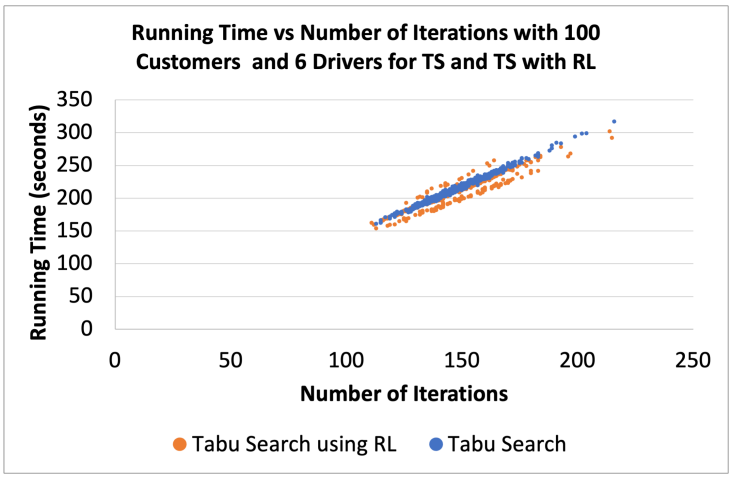
Figure 5.71 Scatter Plot comparing Running Time between RL, TS and TS with RL across samples with 50 customers



(a) 4 Drivers



(b) 5 Drivers



(c) 6 Drivers

Figure 5.72 Scatter Plot comparing Running Time between RL, TS and TS with RL across samples with 100 customers

TS is still very performant in terms of solution quality; as shown in the results, TS is hard to beat, which confirms why it is still being used in literature. However, TS's trade-off is that it can slow down significantly as the problem becomes larger with the addition of customers. Although RL's solution was less optimal, from a waiting time, travel impact time and distance perspective, it provided a quicker solution from an algorithmic complexity point. This shows that RL can provide a scalable solution at the cost of optimality.

The hybrid approach of using RL's output as the initial solution for TS showed minimal performance improvements in solution quality, with only occasional instances where it surpassed TS in terms of waiting time and travel impact time. However, as indicated by the distance metric, this hybrid approach was particularly effective in optimising the routing distances. The explanation for why TS with RL had a limited impact on performance and solution quality compared to TS could have been due to the quality of the initial solution generated by the RL model, as suggested by the RL model results. The initial solution produced by RL put the TS in a region of the solution space with a worse local optimum than the initial solution used in Experiment 1, where the customers were equally distributed. The fact that the TS with RL got stuck within this region indicates that the solution space was not convex.

In this area of research, RL has yet to match the performance of TS in terms of optimality. Although TS excelled in finding high-quality solutions, the trade-off of requiring significant computation time does not justify waiting to obtain a solution, making it less suitable for real-world scenarios with large-scale applications where efficiency is essential.

6 Conclusion

This research explored the challenges of developing a solution that optimises vehicle routing for ride-pooling, which is known to be a computationally challenging problem due to its NP-hard nature. While algorithms such as metaheuristics algorithms are already successful in solving such problems, they face challenges as the complexity and size of the problem increases.

6.1 Revisiting the Aim and Objectives

This research aimed to develop a reinforcement learning algorithm capable of solving the Multi-Vehicle Routing for Ride-Pooling Problem (MVRRPP). The problem determines the optimal set of routes by optimising passenger allocation to vehicles and vehicle routing. The RL algorithm aimed to generate these solutions faster than the traditional metaheuristic methods. To reach this aim, the following objectives were set:

6.1.1 Objective 1: Establishing a benchmark using Tabu Search

The first objective was achieved by implementing a metaheuristic benchmark algorithm using Tabu Search to solve the MVRRPP. For this algorithm, an initial solution containing the initial routes was created by equally distributing the customers to each driver's route in a round-robin fashion. The algorithm used the initial solution as a starting point to iteratively explore neighbourhood solutions within the solution space and find the best candidate. The best candidate cost was compared to the best solution using a cost function, where if the cost was lower, it was replaced as the new best solution. The purpose of the cost function was to minimise the total customers' waiting time and travel time and the total distance travelled by all vehicles. This process was repeated for several iterations where, in each iteration, the best candidate of the previous iteration was used to generate new neighbourhood solutions.

The results of the first objective indicated that each metric, consisting of the waiting time, travel impact time and driver distances increased as the number of customers was added to an instance. With the addition of customers, the driver had to make additional stops to pick up and drop off passengers, leading to a longer waiting time for customers to be picked up by the vehicle. The distance travelled by the vehicle also increases as the vehicle has to visit these extra stops. This also increased the travel impact time since additional stops may be made between a customer's pickup and dropoff location. In contrast, the results showed that these metrics decreased as the number of drivers was added to an instance, indicating that the algorithm successfully

distributed the customers along the drivers to reduce each metric. The results also showed that when using TS to solve instances with larger customers, the neighbourhood became more extensive and complex, requiring more iterations to converge to a local optimum and increasing computational time.

6.1.2 Objective 2: Implement a Reinforcement Learning algorithm for the MVRRPP

Objective 2 was implemented by using reinforcement learning to solve the MVRRPP. This was achieved by utilising the REINFORCE algorithm with a dynamic attention model adapted from literature by Peng et al. [75] to include the constraints defined in MVRRPP.

For this algorithm, the MVRRPP was modelled as RL problem, where the state, action and reward function were defined. The state consisted of the coordinates for the depots, pickup and dropoff locations, the available customers' pickup and dropoff locations, the current driver's origin location (for the current route) and the number of occupied seats for the current vehicle. The action space consisted of the set of possible locations the vehicle (agent) can select from at the next step. The reward function consisted of the cost function, which aimed at minimising the waiting time, travel time, and distance travelled in a solution. A penalty for the number of customers left unpicked in an instance was also added to the reward function.

The RL model used a dynamic attention model with a dynamic encoder-decoder architecture. An encoder was used to encode information of the MVRRPP representation, such as the coordinates of locations, to an embedding. A multi-head attention mechanism was implemented in the encoder to focus the attention of the embeddings on unvisited nodes. The decoder was then implemented to use the encoder's embeddings and the current environment's state as input for the decoder's multi-head attention mechanism, which was responsible for understanding the current state and the relationship between nodes required for determining the vehicles' optimal routes. A mask was created for the decoder's attention mechanism to ensure the agent selected a valid node as its next action. This was done by excluding (masking out) invalid actions that do not adhere to the constraints and rules of the MVRRPP. The decoder was then responsible for generating a probability distribution over the possible actions for the agent to determine which location to select next.

The results from this experiment showed that, similarly to TS, the waiting time, travel impact time and distance increased with the addition of customers. Adding more drivers did not necessarily improve waiting time and travel impact time, unlike TS, suggesting that the RL model might have found a difficulty in balancing the trade-off between waiting time and travel impact time. When evaluating the distance metrics, RL

showed that adding more drivers reduced overall distance, indicating that the RL model successfully optimised the routes based on the proximity of locations and effectively distributed customers across routes to minimise the overall distance travelled. When comparing the performance of RL with that of TS, TS performed better with lower waiting time, travel impact time and distance across all instances. The results showed that while TS was faster when solving smaller problems, such as instances with 20 customers, it struggled with larger problems as the number of customers increased with the TS computation time increasing exponentially compared to that of RL.

6.1.3 Objective 3: Initialising Tabu Search with a solution obtained using Reinforcement Learning

Objective 3 was achieved by using the output solution generated by the RL model as the initial solution for the TS model. The objective aimed to determine whether combining RL with TS made the optimisation process faster and improved the MVRRPP solution.

The results showed similar waiting time and travel impact time performance between TS-with-RL and TS, with only a few instances where TS-with-RL surpassed TS. In contrast, RL exhibited the highest waiting time and travel impact time across the two metrics, indicating challenges in optimising vehicle routing solutions effectively. When evaluating the distance metric, TS with RL outperformed the other models with the least distance travelled across all driver instances with different numbers of customers. TS had a slightly higher distance across instances but was still considerably lower than RL's. The results showed that in terms of running time to generate the optimal solution, TS performed the fastest in smaller problems, with TS-with-RL taking the longest, while RL was the fastest to generate solutions in larger problems. Both TS models showed a substantial increase in computation time as the number of customers increased, suggesting limitations in the efficiency for larger instances.

6.2 Summary of Findings

The RL and TS experiments conducted during this research showed that when solving the MVRRPP, TS effectively found higher-quality solutions in terms of waiting time, travel impact time and total distance travelled compared to RL. This confirms why in literature [1] and [60], TS is still a popular algorithm.

The results demonstrated how the TS implementation successfully explored the solution space by iteratively exploring neighbouring solutions and using a tabu list to avoid revisiting previously visited solutions. This allowed it to search for unexplored

regions of the solution space and find the best possible solutions. The trade-off of TS is its computational complexity, resulting in longer computation times when solving large problem instances consisting of many customers. This becomes problematic in real-world ride-pooling systems where the time to generate a solution in real time is crucial. Although the results indicate that using RL involves a trade-off between solution quality and computation time, it was quicker at finding a solution from an algorithmic complexity point, making it a suitable alternative for finding a scalable solution.

Using TS with RL showed minimal performance gains regarding solution quality, with a few instances where it surpassed TS in terms of waiting time, travel impact time and running time. However, when evaluating distance, TS with RL showed the most effectiveness in performance as it was successful in reducing the overall distance, indicating that the initial solution produced using RL could have focused on optimising distance. The fact that the solution generated by TS-with-RL was not reaching the same solution quality as that of TS indicates that the initial solution generated by RL was in a region of the solution space that had a worse local optimum than the TS implementation containing an initial solution with equally distributed customers between drivers. The TS-with-RL solution being stuck in a region with a worse local optimum suggests that the solution space was non-convex.

Therefore, the choice between TS and RL for MVRRPP depends on the specific requirements of the use-case scenario, balancing the need for high-quality solutions against the available computational time.

6.3 Future Work

While most existent research focuses on using metaheuristic algorithms to solve the VRP, this research demonstrated the possibility of implementing RL to solve a more complex variant of the VRP, known as the MVRRPP, paving the way for future work in this field of research.

Further research may be carried out using alternative RL algorithms to solve the MVRRPP, such as DQN, which utilises Q-learning to estimate the cost function value of taking specific actions in particular states, allowing it to generalise similar states and handle large MVRRPP instances. Additional constraints for the MVRRPP may also be explored, such as passenger constraints, where passengers may specify the departure and arrival times, requiring the algorithm to reconsider which vehicles to pick up first while minimising the other customers' waiting time and travel time.

Another possible research area is adapting the MVRRPP to use Electric Vehicles (EVs) rather than vehicles with an internal combustion engine. This problem would

form part of the Electric Vehicle Routing Problem (EVRP), which includes additional operations such as recharging and minimising the total energy consumption due to the battery limitations that limit EVs' driving range. When EVs are used for ride-pooling, the algorithm must determine if there is enough battery charge for the vehicle to reach the pickup and dropoff locations and return to the origin location. If the vehicle doesn't have enough battery for any of those locations, then a different vehicle is allocated to that customer. The EV must always save additional battery for returning to the origin location, where they can recharge their vehicles. Further research can be carried out to explore the effect of the algorithm when optimising routes for EVs with different battery capacities.

6.4 Final Remarks

As ride-pooling services surged in popularity over the last couple of years, there is a need for algorithms to solve the allocation of vehicles for customers' requests and routing optimisation in real time. In this research, we explored the possibility of using a Reinforcement Learning algorithm to solve the Multi-Vehicle Routing for Ride-Pooling Problem, a combinatorial problem typically solved using metaheuristics. We successfully implemented an algorithm for this problem using the REINFORCE algorithm with a dynamic attention model consisting of a dynamic encoder-decoder architecture that can generate solutions faster than traditional metaheuristic algorithms, allowing it to be implemented in real-world systems. While the solution quality of this approach is still inferior to that of the metaheuristics algorithms such as Tabu Search, generating a solution with Reinforcement Learning is significantly faster as the input size grows, offering a real-time solution for this problem.

References

[1] G. Li and J. Li, "An improved tabu search algorithm for the stochastic vehicle routing problem with soft time windows," *IEEE Access*, vol. 8, pp. 158 115–158 124, 2020. DOI: 10.1109/ACCESS.2020.3020093.

[2] B. Rabbouch, F. Saâdaoui, and R. Mraihi, "Efficient implementation of the genetic algorithm to solve rich vehicle routing problems," *Operational Research*, vol. 21, pp. 1763–1791, 2021. DOI: 10.1007/s12351019005210.

[3] J. M. Vera and A. G. Abad, "Deep reinforcement learning for routing a heterogeneous fleet of vehicles," in *2019 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, 2019, pp. 1–6. DOI: 10.1109/LA-CCI47412.2019.9037042.

[4] K. Zhang, F. He, Z. Zhang, X. Lin, and M. Li, "Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach," *Transportation Research Part C: Emerging Technologies*, vol. 121, p. 102 861, 2020, ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2020.102861.

[5] T. Mustakhov, Y. Akhmetbek, and A. Bogyrbayeva, "Deep reinforcement learning for stochastic dynamic vehicle routing problem," in *2023 17th International Conference on Electronics Computer and Computation (ICECCO)*, 2023, pp. 1–5. DOI: 10.1109/ICECCO58239.2023.10147154.

[6] G. Laporte, "What you should know about the vehicle routing problem," *Naval Research Logistics (NRL)*, vol. 54, no. 8, pp. 811–819, 2007. DOI: https://doi.org/10.1002/nav.20261.

[7] R. Basso, B. Kulcsár, I. Sanchez-Diaz, and X. Qu, "Dynamic stochastic electric vehicle routing with safe reinforcement learning," *Transportation Research Part E: Logistics and Transportation Review*, vol. 157, p. 102 496, 2022, ISSN: 1366-5545. DOI: https://doi.org/10.1016/j.tre.2021.102496.

[8] J. Zhao, M. Mao, X. Zhao, and J. Zou, "A hybrid of deep reinforcement learning and local search for the vehicle routing problems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 11, pp. 7208–7218, 2021. DOI: 10.1109/TITS.2020.3003163.

[9] P. Toth and D. Vigo, *The Vehicle Routing Problem*, P. Toth and D. Vigo, Eds. Society for Industrial and Applied Mathematics, 2002. DOI: 10.1137/1.9780898718515.

[10] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959, ISSN: 00251909, 15265501.

[11] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse, "The vehicle routing problem: State of the art classification and review," *Computers & Industrial Engineering*, vol. 99, pp. 300–313, 2016, ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2015.12.007.

[12] S.-Y. Tan and W.-C. Yeh, "The vehicle routing problem: State-of-the-art classification and review," *Applied Sciences*, vol. 11, no. 21, 2021, ISSN: 2076-3417. DOI: 10.3390/app112110295.

[13] G. D. Konstantakopoulos, S. P. Gayialis, and E. P. Kechagias, "Vehicle routing problem and related algorithms for logistics distribution: A literature review and classification," eng, *Operational research*, vol. 22, no. 3, pp. 2033–2062, 2022, ISSN: 1109-2858.

[14] R. Necula, M. Breaban, and M. Raschip, "Tackling dynamic vehicle routing problem with time windows by means of ant colony system," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, 2017, pp. 2480–2487. DOI: 10.1109/CEC.2017.7969606.

[15] H. Park, D. Son, B. Koo, and B. Jeong, "Waiting strategy for the vehicle routing problem with simultaneous pickup and delivery using genetic algorithm," *Expert Systems with Applications*, vol. 165, p. 113 959, 2021, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.113959.

[16] W. Ho, G. T. Ho, P. Ji, and H. C. Lau, "A hybrid genetic algorithm for the multi-depot vehicle routing problem," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 4, pp. 548–557, 2008, ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2007.06.001.

[17] H. QIN, X. SU, T. REN, and Z. LUO, "A review on the electric vehicle routing problems: Variants and algorithms," eng, *Frontiers of Engineering Management*, vol. 8, no. 3, pp. 370–389, 2021, ISSN: 2095-7513.

[18] S. Karakatič, "Optimizing nonlinear charging times of electric vehicle routing with genetic algorithm," *Expert Systems with Applications*, vol. 164, p. 114 039, 2021, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.114039.

[19] S. Zhang, Y. Gajpal, S. Appadoo, and M. Abdulkader, "Electric vehicle routing problem with recharging stations for minimizing energy consumption," *International Journal of Production Economics*, vol. 203, pp. 404–413, 2018, ISSN: 0925-5273. DOI: https://doi.org/10.1016/j.ijpe.2018.07.016.

[20] M. Mavrovouniotis, C. Li, G. Ellinas, and M. Polycarpou, "Parallel ant colony optimization for the electric vehicle routing problem," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2019, pp. 1660–1667. DOI: 10.1109/SSCI44817.2019.9003153.

[21] T. Dokeroglu, E. Sevinc, T. Kucukyilmaz, and A. Cosar, "A survey on new generation metaheuristic algorithms," *Computers & Industrial Engineering*, vol. 137, p. 106 040, 2019, ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2019.106040.

[22] M. Abdel-Basset, L. Abdel-Fatah, and A. K. Sangaiah, "Metaheuristic algorithms: A comprehensive review," in *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications*, ser. Intelligent Data-Centric Systems, A. K. Sangaiah, M. Sheng, and Z. Zhang, Eds., Academic Press, 2018, pp. 185–231, ISBN: 978-0-12-813314-9. DOI: https://doi.org/10.1016/B978-0-12-813314-9.00010-4.

[23] A. H. Gandomi, X.-S. Yang, S. Talatahari, and A. H. Alavi, *Metaheuristic Applications in Structures and Infrastructures*, 1st ed. Elsevier, Jan. 2013, ISBN: 978-0-12-398364-0.

[24] P. Agrawal, H. F. Abutarboush, T. Ganesh, and A. W. Mohamed, "Metaheuristic algorithms on feature selection: A survey of one decade of research (2009-2019)," *IEEE Access*, vol. 9, pp. 26 766–26 791, 2021. DOI: 10.1109/ACCESS.2021.3056407.

[25] A. Amuthan and K. Deepa Thilak, "Survey on tabu search meta-heuristic optimization," in *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, 2016, pp. 1539–1543. DOI: 10.1109/SCOPES.2016.7955697.

[26] F. Glover and M. Laguna, "Tabu search," in *Handbook of Combinatorial Optimization: Volume1–3*, D.-Z. Du and P. M. Pardalos, Eds. Boston, MA: Springer US, 1998, pp. 2093–2229, ISBN: 978-1-4613-0303-9. DOI: 10.1007/978-1-4613-0303-9_33.

[27] E. Aarts, J. Korst, and W. Michiels, "Simulated annealing," in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. Burke and G. Kendall, Eds. Boston, MA: Springer US, 2014, pp. 265–285, ISBN: 978-1-4614-6940-7. DOI: 10.1007/978-1-4614-6940-7_10.

[28] D. Henderson, S. H. Jacobson, and A. W. Johnson, "The theory and practice of simulated annealing," in *Handbook of Metaheuristics*, F. Glover and G. A. Kochenberger, Eds. Boston, MA: Springer US, 2003, pp. 287–319, ISBN: 978-0-306-48056-0. DOI: 10.1007/0-306-48056-5_10.

[29] Y. Kaya, M. Uyar, and R. Tekjn, *A novel crossover operator for genetic algorithms: Ring crossover*, 2011. arXiv: 1105.0355 [cs.NE].

[30] M. Bramer, *Principles of Data Mining*. Springer London, 2020. DOI: 10.1007/978-1-4471-7493-6.

[31] J. Patterson and A. Gibson, *Deep Learning : A Practitioner's Approach*. O'Reilly Media, Inc., 2017.

[32] J. D. Kelleher, *Deep Learning*. The MIT Press, 2019.

[33] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. DOI: 10.1038/nature14539.

[34] L. Zhang, S. Wang, and B. Liu, *Deep learning for sentiment analysis : A survey*, 2018. arXiv: 1801.07883 [cs.CL].

[35] A. Vaswani *et al.*, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL].

[36] F. Xia *et al.*, "Graph learning: A survey," *IEEE Transactions on Artificial Intelligence*, vol. 2, no. 2, pp. 109–127, 2021. DOI: 10.1109/TAI.2021.3076021.

[37] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021. DOI: 10.1109/TNNLS.2020.2978386.

[38] N. A. Asif *et al.*, "Graph neural network: A comprehensive review on non-euclidean space," *IEEE Access*, vol. 9, pp. 60 588–60 606, 2021. DOI: 10.1109/ACCESS.2021.3071274.

[39] L. Wu, P. Cui, J. Pei, and L. Zhao, *Graph Neural Networks: Foundations, Frontiers, and Applications*, 1st ed. Springer Singapore, Jan. 2022, ISBN: 978-981-16-6053-5. DOI: 10.1007/978-981-16-6054-2.

[40] J. Zhou *et al.*, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020, ISSN: 2666-6510. DOI: https://doi.org/10.1016/j.aiopen.2021.01.001.

[41] G. Wang, R. Ying, J. Huang, and J. Leskovec, *Improving graph attention networks with large margin-based constraints*, 2019. arXiv: 1910.11945 [cs.LG].

[42] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, *Graph attention networks*, 2018. arXiv: 1710.10903 [stat.ML].

[43] M. Naeem, S. T. H. Rizvi, and A. Coronato, "A gentle introduction to reinforcement learning and its application in different fields," *IEEE Access*, vol. 8, pp. 209 320–209 344, 2020. DOI: 10.1109/ACCESS.2020.3038605.

[44] A. Plaat, *Deep Reinforcement Learning*. Singapore: Springer, 2022, ISBN: 9789811906374.

[45] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017. DOI: 10.1109/MSP.2017.2743240.

[46] R. S. Sutton and A. Barto, *Reinforcement learning: An Introduction*, Second edition. Cambridge, Massachusetts ; London, England: The MIT Press, 2018, ISBN: 9780262039246.

[47] P. Ladosz, L. Weng, M. Kim, and H. Oh, "Exploration in deep reinforcement learning: A survey," *Information Fusion*, vol. 85, pp. 1–22, 2022, ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2022.03.003.

[48] A. Alharin, T.-N. Doan, and M. Sartipi, "Reinforcement learning interpretation methods: A survey," *IEEE Access*, vol. 8, pp. 171 058–171 077, 2020. DOI: 10.1109/ACCESS.2020.3023394.

[49] Z. Zhang, D. Zhang, and R. C. Qiu, "Deep reinforcement learning for power system applications: An overview," *CSEE Journal of Power and Energy Systems*, vol. 6, no. 1, pp. 213–225, 2020. DOI: 10.17775/CSEEJPES.2019.00920.

[50] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[51] H. Dong, Z. Ding, and S. Zhang, *Deep Reinforcement Learning Fundamentals, Research and Applications*. Singapore: Springer Singapore, 2020, ISBN: 9789811540950.

[52] R. Nian, J. Liu, and B. Huang, "A review on reinforcement learning: Introduction and applications in industrial process control," *Computers & Chemical Engineering*, vol. 139, p. 106 886, 2020, ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2020.106886.

[53] J. Zhang, J. Kim, B. O'Donoghue, and S. Boyd, *Sample efficient reinforcement learning with reinforce*, 2020. arXiv: 2010.11364 [cs.LG].

[54] F. AlMahamid and K. Grolinger, "Reinforcement learning algorithms: An overview and classification," in *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, IEEE, Sep. 2021. DOI: 10.1109/ccece53047.2021.9569056.

[55] E. Noorani and J. S. Baras, "Risk-sensitive reinforce: A monte carlo policy gradient algorithm for exponential performance criteria," in *2021 60th IEEE Conference on Decision and Control (CDC)*, 2021, pp. 1522–1527. DOI: 10.1109/CDC45484.2021.9683645.

[56] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, pp. 229–256, 1992.

[57] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. DOI: https://doi.org/10.1038/nature14236.

[58] H. van Hasselt, A. Guez, and D. Silver, *Deep reinforcement learning with double q-learning*, 2015. arXiv: 1509.06461 [cs.LG].

[59] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG].

[60] M. Gmira, M. Gendreau, A. Lodi, and J.-Y. Potvin, "Tabu search for the time-dependent vehicle routing problem with time windows on a road network," *European Journal of Operational Research*, vol. 288, no. 1, pp. 129–140, 2021, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2020.05.041.

[61] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin, "A tabu search heuristic for the vehicle routing problem with soft time windows," *Transportation Science*, vol. 31, pp. 170–186, May 1997. DOI: 10.1287/trsc.31.2.170.

[62] M. Polacek, R. F. Hartl, K. Doerner, and M. Reimann, "A variable neighborhood search for the multi depot vehicle routing problem with time windows," *Journal of Heuristics*, vol. 10, no. 6, pp. 613–627, 2004. DOI: https://doi.org/10.1007/s1073200554325.

[63] J.-F. Cordeau, G. Laporte, and A. Mercier, "A unified tabu search heuristic for vehicle routing problems with time windows," *The Journal of the Operational Research Society*, vol. 52, no. 8, pp. 928–936, 2001, ISSN: 01605682, 14769360.

[64] V. F. Yu, H. Susanto, P. Jodiawan, T.-W. Ho, S.-W. Lin, and Y.-T. Huang, "A simulated annealing algorithm for the vehicle routing problem with parcel lockers," *IEEE Access*, vol. 10, pp. 20 764–20 782, 2022. DOI: 10.1109/ACCESS.2022.3152062.

[65] Y. Lin, W. Li, F. Qiu, and H. Xu, "Research on optimization of vehicle routing problem for ride-sharing taxi," *Procedia - Social and Behavioral Sciences*, vol. 43, pp. 494–502, 2012, 8th International Conference on Traffic and Transportation Studies (ICTTS 2012), ISSN: 1877-0428. DOI: https://doi.org/10.1016/j.sbspro.2012.04.122.

[66] W. M. Herbawi and M. Weber, "A genetic and insertion heuristic algorithm for solving the dynamic ridematching problem with time windows," in *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '12, Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 2012, pp. 385–392, ISBN: 9781450311779. DOI: 10.1145/2330163.2330219.

[67] A. O. Al-Abbasi, A. Ghosh, and V. Aggarwal, "Deeppool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 12, pp. 4714–4727, 2019. DOI: 10.1109/TITS.2019.2931830.

[68] G. Guo and Y. Xu, "A deep reinforcement learning approach to ride-sharing vehicle dispatching in autonomous mobility-on-demand systems," *IEEE Intelligent Transportation Systems Magazine*, vol. 14, no. 1, pp. 128–140, 2022. DOI: 10.1109/MITS.2019.2962159.

[69] J. Alonso-Mora, A. Wallar, and D. Rus, "Predictive routing for autonomous mobility-on-demand systems with ride-sharing," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 3583–3590. DOI: 10.1109/IROS.2017.8206203.

[70] A. Wren and A. Holliday, "Computer scheduling of vehicles from one or more depots to a number of delivery points," *Journal of the Operational Research Society*, vol. 23, no. 3, pp. 333–344, 1972. DOI: 10.1057/jors.1972.53.

[71] G. Clarke and J. W. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations research*, vol. 12, no. 4, pp. 568–581, 1964.

[72] W. Kool, H. van Hoof, and M. Welling, *Attention, learn to solve routing problems!* 2019. arXiv: 1803.08475 [stat.ML].

[73] K. Zhang, X. Lin, and M. Li, "Graph attention reinforcement learning with flexible matching policies for multi-depot vehicle routing problems," *Physica A: Statistical Mechanics and its Applications*, vol. 611, p. 128 451, 2023, ISSN: 0378-4371. DOI: https://doi.org/10.1016/j.physa.2023.128451.

[74] A. Gupta, S. Ghosh, and A. 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)*, ser. CODS-COMAD 2022, Bangalore, India: Association for Computing Machinery, 2022, pp. 236–240, ISBN: 9781450385824. DOI: 10.1145/3493700.3493723.

[75] B. Peng, J. Wang, and Z. Zhang, *A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems*, 2020. arXiv: 2002.03282 [cs.LG].

the origin location, and it must return to its origin location after completing its route. Given a set of passenger requests, the objective of the problem is to maximise the utilisation of vehicles and the efficiency of their routes. The problem maximises the utilisation of vehicles by allocating a nearby vehicle to passengers with similar pickup and dropoff locations. The problem then maximises the efficiency of routes by optimising the sequence of pickups and dropoff locations by considering factors such as passenger waiting time and travel time and the distance travelled by the vehicle to minimise costs and environmental impact.

1.2 Motivation

With the growing popularity and demand for ride-pooling services, there is a need to develop algorithms that optimise routes in real-time, as this requires rapid processing of incoming requests and dynamically matching passengers with nearby drivers. Routes must be adapted in real-time to accommodate the addition of customers while minimising detours and waiting times for passengers. By efficiently assigning passengers to vehicles and determining optimal routes in real-time, such algorithms aim to reduce overall travel time, distance travelled, and traffic congestion by combining multiple passengers into one vehicle and, therefore, reducing carbon emissions.

While metaheuristic methods such as Tabu Search [1] and Genetic algorithms [2] have traditionally been used to solve the VRP, these methods often struggle to adapt to complex scenarios with many customers [3]. Metaheuristic methods often find difficulty in solving such problems within an efficient amount of time as the problem increases in demand [4]. However, recent reinforcement learning advancements present a promising approach for tackling VRP more adaptively and efficiently by learning techniques capable of capturing the dynamic and stochastic nature of the VRP [5].

1.3 Challenges

Due to the VRP being an NP-hard combinatorial optimisation problem [6], the number of possible solutions increases exponentially as the number of vehicles and passengers increases, encountering issues such as solution quality and slower running times when using metaheuristic methods [7, 8]. Recent advancements in Reinforcement Learning (RL) have shown impressive performance in solving combinational optimisation problems due to their ability to learn optimal decision-making policies through interactions with the problem environment [5].

However, solving the MVRRPP using RL poses several challenges. Such

challenges include defining the appropriate state representation for the MVRRPP in a way that allows the RL agent to capture the essential features such as the location of drivers and customers, vehicle capacity, distance travelled and waiting time, allowing it to learn how to optimise vehicle allocation and routing.

Implementing an RL algorithm with a dynamic attention model is also challenging as it involves designing a dynamic encoder-decoder architecture with multi-head attention layers. Such a model aims to implement an encoder that can selectively focus on different parts of the input data, such as the remaining locations for pickup and dropoffs. It also aims at implementing a decoder that can focus on relevant features of the encoded input representation, such as distance between locations, to help the RL decide which customers to visit next.

Unlike the traditional VRP, the MVRRPP includes additional constraints related to the ride-pooling aspect of the problem. Such constraints include visiting a dropoff location before a pickup location, visiting a visited location or picking up additional customers when the maximum seating capacity of the vehicle is reached. Implementing these constraints is a significant challenge as the RL agent must adhere to such constraints.

Another challenge of this research is finding the appropriate reward function that guides the RL agent during training to find the optimal or near-optimal solutions when solving the MVRRPP. The MVRRPP consists of a multi-objective function used to measure the solution's cost and quality. These objectives include minimising passenger waiting time and travel time and the total distance travelled by the vehicles. Achieving a balance among these objectives is challenging since improving one objective may result in a trade-off with another.

1.4 Aims and Objectives

The aim of this research is to develop an algorithm that uses reinforcement learning to solve the ride-pooling problem and generate high-quality solutions faster than the traditional metaheuristic methods. The algorithm will optimise the allocation of passengers to vehicles, depending on their location and destination, and optimise vehicle routing by minimising the overall waiting time, travel time and total driving distance. In order to reach the stated aim, the following objectives have been set:

1. Implement the Multi-Vehicle Routing for Ride-Pooling Problem (MVRRPP) using Tabu Search as a benchmark algorithm and evaluate its performance in terms of customer's waiting time, travel impact time, distance travelled and time to generate the solution.

2. Model the MVRRPP as a Reinforcement Learning (RL) problem by finding the appropriate representation of states and actions and defining a reward function that balances the trade-off between minimising customers' waiting and travel time, and finally compare RL's performance with Tabu Search.

3. Evaluate the effect of combining a Reinforcement Learning approach with optimisation based on Tabu Search and compare the performance with that of Tabu Search and RL on their own.

1.5 Proposed Solution

To accomplish the objectives stated in Section 1.4, three experiments will be implemented. The first objective will be achieved by implementing a ride-pooling service using tabu search, an algorithm widely used in research to solve the VRP. In this experiment, the algorithm's initial solution will be generated by distributing the customers equally to each driver's route. The initial solution will then be used by the algorithm to iteratively explore and refine the solution space until an optimal solution is found. A solution is considered optimal if it cannot minimise the waiting time, travel time, and distance travelled any further.

The second objective will be achieved by implementing a reinforcement learning algorithm called REINFORCE, a policy gradient algorithm with a baseline model that aims to reduce the variance of the gradient estimates. In this experiment, the MVRRPP will be modelled as an RL problem in which the state will consist of information on the current environment, such as remaining customers, the driver's starting location, and vehicle occupancy and the actions will consist of selecting a location. The reward function will consist of minimising the customer's waiting time, travel time, and the distance travelled by the vehicle while also minimising the number of remaining customers left unpicked at the end of a MVRRPP instance. A dynamic attention model with a dynamic encoder-decoder architecture will be implemented, where the encoder will be used to process the location coordinates and create representation embeddings, and the decoder will be used to generate the output sequence incrementally by selecting locations (actions) at each step based on a probability distribution.

The third objective will be addressed by using the output solution generated from the second experiment as an initial solution for the tabu search algorithm. This experiment aims to determine whether reinforcement learning can improve the solution quality in the tabu search by bootstrapping it with a different initial solution.

Each experiment will evaluate the MVRRPP on various scenarios, encompassing different numbers of customers and drivers. For each scenario containing 4, 5, and 6 drivers, scenarios with 20, 50 and 100 customers will be created. A dataset will be

created for each scenario, containing 500 different MVRRPP instances. Evaluating each experiment in different scenarios helps determine performance on larger problem instances.

1.6 Contributions

While prior research focuses on solving the VRP for logistics purposes, this research aimed to solve the MVRRPP, a variant of the VRP, with the additional complexities related to ride-pooling. This problem required dynamic and real-time decision-making based on continuously changing data. As a customer requests a ride, the algorithm has to dynamically assign a vehicle to that customer and update the vehicle's route to include the new customer's pickup and dropoff location. This research adapted the REINFORCE algorithm with a dynamic attention model to solve the MVRRPP. Using a dynamic encoder-decoder architecture, the model efficiently generated the routes for multiple vehicles serving passengers with similar destinations.

The model's performance was compared with Tabu Search (TS), a metaheuristic algorithm often used to solve the VRP. Evaluating the MVRRPP using TS proved to be effective, achieving the highest quality solutions compared to the RL model. A hybrid approach was also implemented that combines RL with TS to optimise RL's solution further. The solution achieved using the RL model was used as an initial solution for the TS to begin its search process. This approach was only effective in reducing the distance travelled.

The overall results showed that when solving larger MVRRPP instances, the RL model achieved quicker solutions than metaheuristic methods but with a trade-off in solution quality. This demonstrates how the RL model can be applied to real-world ride-pooling scenarios, as it can produce a scalable solution within reasonable execution times.

1.7 Document Structure

The rest of this document is structured as follows:

Chapter 2: Background

Provides background information on the study's relevant research areas. It will include an overview of the vehicle routing problem, metaheuristic algorithms, deep-learning architectures such as neural networks, deep learning, graph neural networks, transformers, and reinforcement learning.

Chapter 3: Literature Review

It reviews existing research comprehensively, including solving different variants of the vehicle routing problem using metaheuristic algorithms and reinforcement learning approaches.

Chapter 4: Methodology

Comprises the design and implementation of three experiments. It first describes the formulation of the MVVRP and its cost function, followed by a detailed description of the techniques used to implement tabu search, reinforcement learning, and the hybrid approach that uses tabu search with reinforcement learning.

Chapter 5: Results & Evaluation

Evaluates the results of the experiments outlined in the methodology chapter. The performance of each experiment was evaluated using the solution running time and the cost function, which consisted of waiting time, travel time, and distance.

Chapter 6: Conclusion

This chapter revisits the aim and objectives and discusses what was achieved. A summary of the findings obtained in the results chapter is given, followed by possible future work and, some the final remarks.

2 Background

This chapter provides the essential background information to understand the problem domain and presents a theoretical framework for the discussed techniques. The research covered in this chapter includes background information on the vehicle routing problem and its variants, metaheuristic techniques, and background information on deep-learning architectures such as neural networks, deep learning, graph neural networks, transformers, and reinforcement learning.

2.1 The Vehicle Routing Problem

The VRP aims to find efficient routes for vehicles fleets that provides a service within a specified duration to a group of customers. Services include delivering goods and transportation systems [9]. The first introduction of the VRP was by Dantzig and Ramser [10], where they formulated the problem as the 'Truck Dispatching Problem'. This problem consisted of finding the shortest routes for a group of gasoline delivery trucks from a central terminal to different service stations while satisfying oil demands and minimizing the total distance travelled by the delivery trucks. Using a linear programming formulation, they propose the first mathematical formula for determining a nearly optimal solution.

Recent VRP models have evolved significantly from the original formulation of the problem to cater for real-world scenarios, such as time windows for delivery and pickup, traffic conditions affecting travel time durations, and demand which changes dynamically through time [11].

2.1.1 Problem Notations

The VRP can be defined as a graph in which the vertices represent the different locations and road junctions while the arcs represent the roads. The graphs and their corresponding arcs can either be directed or undirected, depending on whether they allow movement in only one direction (such as one-way streets) or in both directions. Every arc has a cost, which contains its distance and duration. The type of vehicle and timeframe when the arc is traversed may influence the cost [9].

When defining the VRP on an undirected graph, $G = (V, A)$, where $V = \{0, 1, ..., n\}$ represents the vertex set and $A = \{\{i, j\} : i, j \in V, i \neq j\}$ represents the arc sets. One of the vertices (Vertex 0) represents the depot, which includes m identical vehicles, each at capacity Q. A non-negative demand $q_i \leq Q$ is allocated to each customer $i \in V \setminus \{0\}$ (excluding the depot) [6].

The solution to the VRP can be defined as the total costs linked to the arcs, where each arc represents a vehicle route. The objective of the VRP is to find the set of routes that minimise the costs. Each route is assigned to a single vehicle and contains the sequence of vertices the vehicle visits [9]. Each vehicle must start and finish each route at the depot and not exceed its maximum capacity, Q. Each customer location must be visited by one vehicle, where the customer's demands are satisfied in that visit and not split across multiple visits by that vehicle [12].

2.1.2 Extensions and Variants

Different VRP variants were introduced in research to cater for the requirements and constraints in real-life routing problems. Examples of these constraints include the chosen routes, duration, and length. Each variant's objective is to find the optimal solution to the problem while minimising overall costs [13]. Some of the most researched VRP variants are listed below.

The VRP with time windows (VRPTW) involves the addition of time window restrictions associated with each customer. The vehicle must serve a customer i during the specified time interval $[e_i, l_i]$, where e_i represents the earlier arrival time whilst l_i represents the latest arrival time. The vehicle can only visit the customer i after the earliest arrival time e_i. The VRPTW also includes a service time that represents the time the vehicle takes to serve a customer [14].

The VRP with simultaneous pickup and delivery (VRPSPD) is another variant of the VRP, which allows vehicles to perform concurrent pickups and deliveries during a route. Each vehicle starts by loading the items for delivery at the depot and finishes by returning any collected items to the depot. When visiting a customer, the vehicle can simultaneously carry out delivery and pickup demands while ensuring it is within its load capacity when picking additional items. The VRPSPD aims to establish a series of routes that uses a limited amount of vehicles to satisfy the customer's demand while staying within a maximum travel distance [15].

The Multiple Depots VRP (MDVRP) involves several depots from which vehicles can depart and serve customers. The MDVRP requires that one vehicle visits each customer and that each vehicle begins and finishes its route from the same depot. The first stage of MDVRP consists of determining which depot to assign to a group of customers based on their distance to that depot. In the second stage, customers from the same depot are allocated to multiple routes while ensuring the vehicle is within its maximum capacity. The last stage involves determining each route's sequence for the vehicle to serve the customers. Similarly to other variants, the MDVRP objective is to reduce the overall delivery time (by minimising the number of routes) and operation costs (by minimising the number of vehicles used) [16].

The Electric Vehicle Routing Problem (EVRP) extends on the VRP to include electric vehicles (EVs) and operations such as recharging and minimising the total energy consumption [17]. Due to EVs' battery constraints, EVs have a shorter driving range than vehicles with an internal combustion engine, which restricts their overall driving distance. When planning longer routes, the charging of EVs at a charging station must be accounted for [18]. The EVRP aims to find the most efficient routes that minimise the EVs' energy consumption while considering constraints such as the battery capacity and limited availability of charging stations (CSs) within the area. If their battery energy gets depleted, EVs can visit CSs en route, including the depot where the EV battery can get recharged. Several factors, such as the speed, travel distance and overall vehicle load, may affect the EV's battery consumption [19]. Companies such as those in the logistics distribution sector are utilising EVs in their day-to-day operations to reduce their carbon footprint. The EVRP enables them to determine the most efficient routes for a fleet of EVs to serve a collection of customers while evaluating the constraints associated with electric vehicles [20].

2.1.3 Multi-Vehicle Routing for Ride-Pooling Problem

This research investigates the MVRRPP. This problem involves finding the most efficient routes for driver vehicles to pick up and drop off a set of passengers while minimising costs. Each driver and customer are located in different geographical locations. Each passenger has a pickup location and a drop-off location. Vehicles must first visit passengers' pickup locations before visiting their corresponding drop-off locations. When picking up customers, a driver must start and end a route from the same location and should remain within its seating capacity. The objective of the problem is to minimise the total distance travelled by vehicles while reducing passengers' waiting time and travel time. The waiting time consists of the time taken for the vehicle to pick up a passenger, whilst the travel time consists of the extra time a passenger spends inside the vehicle between pickup and dropoff due to ride-sharing, where the driver may pick up or drop off other customers.

2.2 Metaheuristic Algorithms

Metaheuristics are problem-solving techniques that operate on a high level to tackle various optimisation problems [21]. Optimisation problems, such as those that are NP-Hard, are too complex to search for every possible solution, making it impossible to find the exact optimal solution. Metaheuristic algorithms aim to efficiently find acceptable solutions for optimisation problems in a feasible time [22].

Metaheuristic algorithms balance local exploration and global search, leveraging randomization to achieve diverse solutions. Randomization enables metaheuristic algorithms to explore beyond local search boundaries and search a broader, global scale of potential solutions, making them well-suited for global optimisation and nonlinear modelling. The core elements of every metaheuristic algorithm are exploitation (intensification) and exploration (diversification). Exploration involves creating different solutions to explore a broader search space on a global scale, whilst exploitation involves using the information obtained from a valid solution in the local region to explore search within that specific region. A well-balanced combination of exploitation and exploration typically enhances the likelihood of selecting the best solutions that converge to optimality and attaining the global solution [23].

2.2.1 Metaheuristics Techniques

Metaheuristic algorithms fall into two primary groups: single solution-based metaheuristics and population solution-based metaheuristics. Single solution-based metaheuristics begin the optimisation process using a single solution that gets modified and refined throughout each iterative step. These algorithms could, however, get trapped at local optima and might not extensively explore the entire search space. In contrast, population-based metaheuristics start by creating a set (population) of solutions, which are then updated iteratively. By leveraging multiple solutions, these algorithms could explore more regions of the search space and increase the likelihood of escaping a local optimum. Metaheuristic algorithms can further be grouped into four categories based on their behaviour: swarm intelligence-based, evolution-based, physics-based and human-related algorithms. Swarm intelligence algorithms derive from the social behaviours of animals, insects, birds, or fish. Such algorithms include Particle Swarm Optimisation (PSO) and Ant Colony optimisation. Evolution-based algorithms derive from natural evolution, where a population of solutions are randomly generated at the start, and the best solutions are selected to create new individuals. Such algorithms include the Genetic algorithm and Tabu Search. Physics-based algorithms derive inspiration from the principles governing the universe's physics, such as Simulated Annealing, whilst human behaviour-related algorithms derive inspiration from human behaviour, such as the Teaching learning-based optimisation algorithm (TLBO) [24].

2.2.2 Types of Metaheuristic Algorithms

Tabu Search

Tabu Search (TS) is a local search metaheuristic algorithm that aims to solve optimisation problems by utilising exploration strategies and adaptable memory approaches that help it avoid local optima [25]. TS uses a tabu list to store past selected solutions, enabling the search to avoid reselecting the same solutions and explore alternative solutions during the search process [22]. TS may employ different types of memory structures. Such memory structures include short-term, which involves storing the most recent list of solutions; intermediate-term, which implements intensification rules to focus the search on promising regions within the search space; and long-term, which focuses on diversification rules to guide the exploration towards new regions of the search space. In TS, the algorithm starts from an initial solution and generates neighbour solutions from that initial solution. It then uses the objective function to find the best solution from the set of neighbours [25]. The neighbour solutions are generated by applying a move mechanism, such as swapping and replacing elements within the current solution to create other potential solutions for the neighbourhood [26].

Simulated Annealing

Simulated Annealing (SA) is a stochastic local search method for solving combinatorial optimisation problems. SA draws significant inspiration from analogising the physical process of annealing in solids and solving combinatorial optimisation problems. Annealing in condensed matter physics represents the thermal technique used to acquire low-energy states of a solid within a heat bath [27]. Using this analogy for SA, the energy state represents the SA's objective function, and the shift in the energy state represents the transition from the current solution to a neighbouring one [23]. SA accepts one of two types of solutions in each iteration: an improved solution and a non-improving solution. By accepting 'hill climbing' moves (worse solutions), SA avoids local optima when searching for a global optimum. The algorithm's temperature parameter determines the probability that the algorithm accepts these non-improved solutions. The 'hill climbing' moves occur less often as the temperature parameter decreases, causing the probability of accepting worse solutions to decrease, allowing the algorithm to converge towards better solutions. The algorithm may still allow for occasional exploration of less optimal solutions to avoid getting trapped in the local optima [28].

Genetic Algorithms

Genetic Algorithms (GA) are stochastic search techniques inspired by genetics and natural selection principles. GA encode the search problem's decision variables into finite-length strings that represent the candidate solutions and are composed of specific characters. These strings are called chromosomes, while the specific characters are called genes, and their values are called alleles. These methods have been applied to the Traveling Salesman Problems (TSP), where, in this case, the routes are the chromosomes, the city is the gene, and the allele is the value representing the specific city [27].

GA begin the search process by randomly creating an initial population of candidate solutions. The objective function computes the fitness of these solutions to prioritise the individuals better suited for reproducing and propagating the next generation (iteration). The individuals (parents) chosen to reproduce the next generation of children are combined through a process known as crossover [23]. The crossover aims to create chromosomes (children) that could outperform both parents by inheriting the best traits from each [29]. A new generation with improved fitness is formed by removing the parents and replacing them with the children. Mutation is periodically incorporated into the population to avoid reaching a local optimum and to encourage exploration within the parameter space. The greater the number of iterations in GA, the higher the likelihood of it converging to a population containing identical members where the optimal solution would have been reached [23].

2.3 Deep-Learning Architectures

2.3.1 Neural Networks

Neural networks are computational systems that draw inspiration from the structure of the brain's neurons and their interconnections. Neural networks consist of neurons organized into a structured sequence of at least three layers: the first layer consists of the input layer, the middle layer consists of the hidden layers, and the last layer is the output layer. This neural network's structure, organized layer by layer, is commonly known as a feed-forward network. During a process called forward propagation, data is sequentially transmitted from the input layer and passed through each hidden layer before reaching the output layer. The connections between nodes are called weights, consisting of numerical values that may be positive or negative [30]. Figure 2.1 illustrates a multilayer feed-forward neural network, where each neuron, represented through a circle, is connected to all the other neurons in its neighbouring layers.

An activation function is applied during forward propagation, which uses the

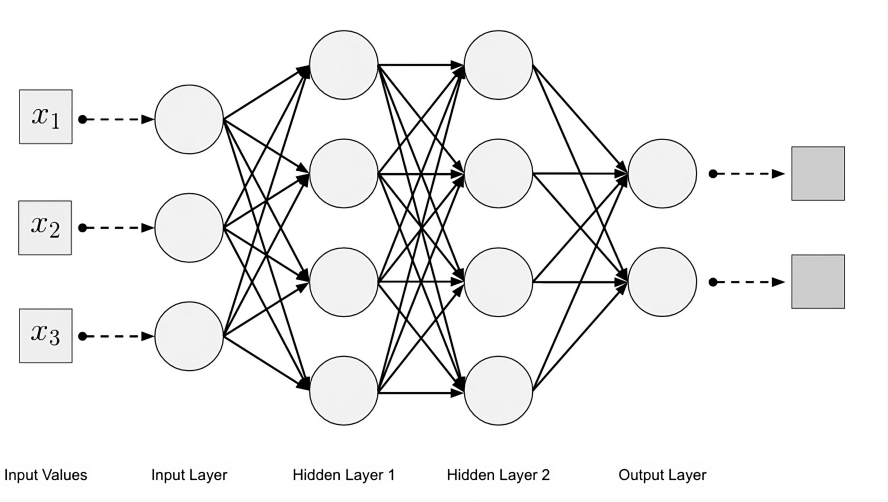


Figure 2.1 Multilayer Feed-Forward Neural Network (Extracted from: Patterson and Gibson [31])

inputs, weights, and biases to calculate the neuron's output value. This output value is then used as input for the next layer. Activation functions determine whether neurons are 'activated'. If the output of the activation function exceeds a certain threshold, typically zero, the neuron is activated [31]. The activation function can be represented using the following equation [32]:

$$f\left(b + \left(\sum_{i=1}^{n} w_i \; x_i\right)\right) \tag{2.1}$$

where $f()$ is the activation function such as tanh or ReLU. Training a feed-forward neural network involves utilizing a learning algorithm such as backpropagation. The backpropagation algorithm is used to estimate the gradient of the loss with respect to the model's parameters (weights). An optimisation algorithm then uses these gradient estimates to compute parameter updates [31].

2.3.2 Deep Learning

Deep learning is a subcategory in machine learning that uses neural networks to enable computational models constructed with multiple layers of processing to acquire knowledge of representations in data with varying levels of abstraction. During deep learning, multiple levels of representations are created by combining non-linear components. Each component is responsible for converting the representation at its current level, beginning with the initial raw input, into a more abstract representation at the next higher level [33]. The initial layers, situated near the data input, acquire knowledge of the basic features, whereas the upper layers acquire knowledge of the more complex features using the data from the lower-level features. This structure

provides a detailed hierarchy of the representation of features [34].

Deep learning uses the backpropagation algorithm to identify complex patterns within large datasets by instructing the network on which parameters to modify. These parameters are responsible for computing each layer's representation based on the preceding layer's representation. Deep learning can also detect complex patterns within data in high-dimensional spaces. This has led to significant breakthroughs in addressing challenges that have long been difficult for other machine learning algorithms [33]. Such breakthroughs include areas in computer vision, natural language processing, speech recognition, and science areas such as medicine [32].

2.3.3 Transformers

Vaswani et al. [35] introduced the Transformer, a network architecture that relies on attention mechanisms for computing input and output representations, eliminating the need for recurrent and convolutional layers. The Transformer uses a stacked self-attention and point-wise with fully connected layers in both the encoder and decoder. The encoder consists of N identical layers, each consisting of a sub-layer comprising a multi-head self-attention mechanism and a fully connected feed-forward network. Each layer has a residual connection with layer normalisation. Similar to the encoder, the decoder also includes N identical layers with an additional sub-layer responsible for performing multi-head attention on the encoder's output stack. The decoder also includes masking on the self-attention sub-layer, restricting positions from attending to subsequent positions in the sequence.

An attention function maps a query vector and a collection of key-value pair vectors to generate an output. It is calculated as a weighted sum that uses the query's compatibility function with its corresponding key to allocate the weight to each value. Transformers utilise scaled dot-product attention for the attention function, which calculates the dot product between the query and all the keys, which is then divided by $\sqrt{d_k}$ and then applied to a softmax function, which determines the weights assigned to the values. The outputs are calculated as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.2}$$

where matrix Q represents the set of queries, matrix K represent the keys and V represent the values. The Multi-head attention layer uses multiple attention layers (heads) that allow the model to focus on different aspects of the sequence in parallel. Each attention head attends to different learned representation subspaces.

2.3.4 Graph Neural Networks

Graph data, which consists of a set of vertices and edges, is often used to depict the network structure of interconnected data. The vertices represent the entities in graph data, while the edges depict the relationship between these entities [36]. While traditional deep learning algorithms have accomplished significant achievements in identifying complex patterns in Euclidean data (like images), several other applications require a graph-based structure to represent data more effectively. An example is e-commerce, where a graph-based learning system can utilise user-product interactions to generate potential recommendations. Graphs are also used in chemistry to model molecules as graphs and identify new drugs based on their bioactivity [37].

Graph learning involves mapping graph features into feature vectors while keeping the exact dimensions within the embedding space. Without reducing the graph's data to a lower dimension, a graph learning model can transform the graph data into the output of a graph learning architecture while identifying the complex relationships between vertices. Many graph learning approaches extend deep learning methods to represent graph data as vectors [36]. An example of a deep learning architecture that works with graph data is Graph Neural Networks (GNN). The GNN's architecture involves passing the input graph to the hidden nodes to acquire knowledge of the representations in the graph data [38].

GNNs can generate outputs for various graph analysis tasks using the graph structure and node content details as input. The GNN can generate outputs at node, edge, and graph levels. The output generated at a node level corresponds to tasks involving node regression and classification, such as ConvGNNs and RecGNNs [37]. A GNN updates the node representations by combining their representation from the prior iteration with the representations of neighbouring nodes [39]. The representations are established by the relationships between vertices using the shared weighted connections [38]. Once the GNN learns the representations of the nodes, the GNN aims to categorise nodes into predetermined classes through node classification [39]. Output tasks at the edge level involve link prediction, which consists of anticipating the presence of an edge between two specified nodes and edge classification, which involves categorising edge types [40]. On a graph level, the output tasks involve graph classification, where the GNN carries out readout and pooling operations to learn graph representations [37].

A Graph Attention Network (GAT) is a GNN architecture that utilises an attention mechanism (inspired by natural language processing) for learning graph representations [41]. By concentrating on the most relevant inputs, attention mechanisms can create a representation for a singular sequence of varying sizes using a self-attention strategy. Using this strategy, the attention-based architecture in the

GATs can calculate the hidden representations of every node. The masked self-attentional stacked layers architecture allows nodes to attend to their neighbourhoods' features by assigning different weights among different nodes within the same neighbourhood without requiring prior knowledge of the graph structure [42].

Attention-based GNNs such as GAT tend to encounter issues with over-smoothing and overfitting. Overfitting may occur with the learned attention functions due to the masked self-attention mechanism restricting the computation of attention weights for only direct neighbours. Over-smoothing may occur when the connected nodes on opposing sides of the class boundary exchange information through the edges. When the multiple attention layers are stacked, these node features may become overly smoothed, making them indistinguishable [41].

2.4 Reinforcement Learning

RL is a machine learning technique where an agent utilises trial and error to learn how to interact in an environment and achieve a particular objective. The agent receives a positive or negative reward based on interactions with the environment. The agent considers the reward as feedback for the actions taken and uses it to further its knowledge of the environment and improve its decision-making. The agent can decide whether to take actions based on existing knowledge of the environment or experiment with actions it has never attempted in that particular scenario [43]. RL aims to find the optimal policy function that provides the best action for any given state. The agent determines the optimal policy through repeated actions within the environment and maximising the cumulative future rewards [44].

Figure 2.2 depicts the RL agent's interaction with the environment. The agent (controlled by the algorithm) at time step t receives from the environment, state s_t. The agent then interacts with the environment by carrying out action a_t, which results in the environment transitioning to a new state, s_{t+1} in the following time step. The state consists of the essential information from the environment for the agent to take the optimal action. When transitioning to the following state, the environment sends feedback to the agent as a scalar reward r_{t+1}. The agent aims to discover a policy π that maximises the cumulative discounted reward. The agent can improve its policy by using knowledge from each state transition $(s_t, a_t, s_{t+1}, r_{t+1})$ [45].

2.4.1 Markov Decision Processes

In RL, the relationship between the learning agent and the environment is defined using the Markov Decision Processes (MDP). We establish this relationship using actions, states and rewards. The MDP represents a conventional approach to

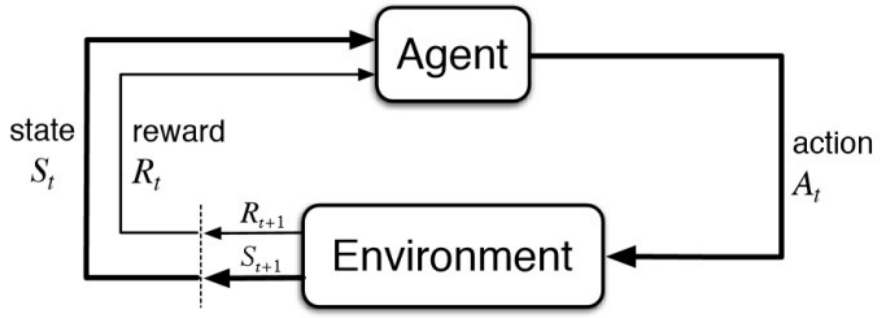


Figure 2.2 The RL's agent interaction within the environment. (Extracted from: Sutton and Barto [46])

formalising sequential decision-making problems. The actions taken in these problems impact not only the current rewards received but also the future states, which in turn affect future rewards [46].

A finite MDP consists of a set of states S, a set of actions A, and a probability distribution $p(s', r\,|s, a)$, where $\{s, s'\} \subseteq S$, $a \in A$, and $r \in \mathbb{R}$, that defines the probability of transiting to state s' from s and obtaining the reward r when applying action a . An MDP can also have the discount factor $\gamma \in [0, 1]$ [46]. The Markov property establishes that the transition function is dependent on information from the current state, selected action, and the resulting state and is independent of past actions and states. Elements like the rules and physics of the environment are stationary and remain consistent. An example is chess, where the player does not need to recall past moves to play his next move [43].

2.4.2 Policy and Value function

At each time step t, the agent obtains states $s_t \in \mathcal{S}$ from the environment, where it then chooses the actions $a_t \in \mathcal{A}$ based on the policy $\pi(a_t|s_t)$. The policy maps the states s_t with the actions a_t. When the agent performs an action a_t, it receives a reward $r_t \in \mathcal{R}$ from the environment. In each state s_t, the agent aims to maximise the cumulative discounted reward $G_t = \sum_{k=0}^{\infty} \gamma^k r_{k+t}$ [47].

The value function $V(s)$ determines how beneficial it is for the agent to be in a particular state while following a given policy π by estimating the value of G_t corresponding to other states. Different policies can provide different values for identical states [48]. This state-value function relies on the specific policy π that the agent adheres to [49]:

$$V^{\pi}(s_t) = \mathbb{E}\left[R_t | s_t = s\right] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right] \tag{2.3}$$

The action-value function Q can be described as the valuation of the outcome when action a is chosen within state s, and then the agent starts following policy π:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}\left[R_t | s_t = s, a_t = a\right] \tag{2.4}$$

$$= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right] \tag{2.5}$$

The optimal policy π^* refers to a policy that acquires the highest cumulative reward over an extended period $\pi^* = \underset{\pi}{\text{argmax}}\, V^{\pi}(s)$. Derived from the Bellman optimality equation [50], Equation (2.6) represents the optimal policy using the state-value function while Equation (2.7) represents the optimal policy using the action-value function.

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{2.6}$$

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a) \tag{2.7}$$

2.4.3 Reinforcement Learning Algorithms

RL algorithms can be categorised as model-free and model-based methods. Model-free methods use trial and error to expand their knowledge of the environment and update the policy accordingly to achieve optimal rewards. Since model-free methods lack information about the transition model and reward function, they need to interact with the environment repeatedly to understand the system's behaviour. Examples of model-free RL are Temporal Difference (TD) and Monte Carlo (MC). In model-based methods, the agent does not depend on experimentation, unlike model-free methods, but instead utilises a predefined learned model to forecast various states and their corresponding rewards. An example of a Model-based RL algorithm is dynamic programming [43].

RL can also be categorised as value-based or policy-based. In value-based RL, the value function is calculated for each state and used to evaluate the policy [48]. In policy-based RL, the policy is updated and optimised directly through each iteration

until it maximises the cumulative return [51]. Actor-critic algorithms combine both value-based and policy-based methods. The critic computes the value function, and the actor adjusts the policy based on the values of the critic [48].

Dynamic Programming (DP) consists of algorithms that use a model of the environment (represented as an MDP) to find an optimal policy [46]. Since DP algorithms are model-based, they require both the transition model and the reward function. Applying DP requires the problem to include the following properties: optimal substructure and overlapping sub-problems. The optimal substructure refers to the property where the best solution for the problem can be deconstructed into solutions for its smaller sub-problems. The overlapping sub-problem property is when the finite sub-problems are encountered recursively, allowing their respective solutions to be stored for potential reuse [51]. Two widely utilised techniques in DP are value iteration and policy iteration. Policy iteration aims to find the optimal policy by iterating through different policies and retaining only the policy that delivers the most significant overall rewards. The optimal policy is determined when the cumulative rewards of π converge. In contrast, instead of evaluating multiple policies, value iteration determines the optimal policy by finding the optimal value functions and sequentially navigating through each state to determine the actions associated with the highest values [52].

MC methods are model-free RL, which rely on learning through the experience they acquire when interacting within an environment (without prior knowledge). Unlike DP, which requires all probability distributions of the transitions, MC methods only require a sample of states, actions and their rewards. MC methods can estimate the state-value function for a given policy π, where the expected returns from the policy are averaged when visiting a state multiple times until the average result converges to the expected value [46]. To estimate the state-value function $V_\pi(s)$ when following policy π, we first gather a set of episodes traversing through state s. Each occurrence of state s in an episode is called a 'visit'. MC methods can then perform the estimations using the first-visit MC or every-visit MC method. The first-visit MC calculates the average estimation in the episode by using only the returns of the first visit to state s. In contrast, the every-visit MC method calculates the average estimation in the episode using all visits to state s [51].

TD is another model-less RL consisting of a combination of concepts from DP and MC. Like MC, TD learns from the experience it acquires when interacting with the environment without requiring an existing model of the environment. When compared to MC, TD allows finding the return after the first time step, while MD requires completing the episode to obtain the return. Another distinction with MC is that when carrying out experimental actions, MC tends to discount or ignore episodes, which can hinder learning; in contrast, TD extracts knowledge from every transition, regardless of their following actions, making them less prone to such challenges [46].

Similarly to DP, TD uses bootstrapping, which involves using the previously calculated value functions to estimate the current value function. Two widely used TD methods are SARSA and Q-learning. Q-learning is an off-policy algorithm where the agent follows an equal probability policy. This policy ensures that all available actions in all states have an equal chance of being selected during training. This approach allows the agent to explore the environment more thoroughly and change to an optimal policy. In contrast, SARSA uses an on-policy algorithm where the policy it follows during training (behaviour policy) is the same as the policy it aims to discover, the target policy. This target policy is assumed to be the optimal policy. The agent is considered on-policy if the target and behaviour policies are the same [52].

2.4.4 Policy-based Methods

Policy-based methods involve learning a parameterised policy that directly selects actions without utilising a value function. Some policy-based approaches may employ a value function to train the policy parameters but do not use it to determine actions [44]. In contrast to value-based RL, a policy-based approach offers improved convergence, simpler policy parameterisation and is better suited when working with continuous or high-dimensional action spaces [51].

Policy-based methods use gradient-free or gradient-based optimisation techniques to update the optimal policy. Gradient-free techniques use heuristic search to find better policies. Such methods consist of evolution strategies, such as hill climbing within a subset of policies. Although gradient-free techniques can efficiently explore low-dimensional parameter spaces, they might not scale efficiently to larger, more complex models. In gradient-based methods, the gradients provide valuable feedback on how to enhance a parameterised policy. In policy gradient RL, deterministic and stochastic approximations are used to find the average of the plausible trajectories (generated by the current policy parameterisation) and then calculate the expected return. Despite their computational demands, gradient-based methods remain the preferred choice due to their efficiency when working with policies with many parameters [45].

REINFORCE is a policy gradient algorithm that uses a single trajectory or a set-sized mini-batch of trajectories with a gradient estimator to calculate a stochastic estimated gradient [53]. REINFORCE stands for **RE**ward **I**ncrement = **N**onnegative **F**actor x **O**ffset **R**einforcement x **C**haracteristic **E**ligibility [54]. The REINFORCE algorithm utilises Monte Carlo techniques to sample episodes and approximate the gradient of the cumulative reward. While providing unbiased estimations, these estimations may experience high variance, increase sample complexity, and impact the learning rate [55]. Without altering the bias, the variance in these estimations may be

reduced by implementing a baseline to the REINFORCE algorithm. The bias is maintained by subtracting the baseline value from the expected return [54]. The REINFORCE algorithm can adapt to different neural network architectures that provide arbitrary outputs, allowing it to be applied to various problem domains. It also offers the flexibility to integrate with other gradient methods, such as backpropagation [56].

2.4.5 Deep Reinforcement Learning

In contrast to traditional RL, which uses analytical approaches to determine the function approximation, Deep Reinforcement Learning (DRL) utilizes deep neural networks, allowing it to work with high dimensional problems [51]. DRL uses deep learning techniques to extract observation data from the environment. This observation data helps in understanding the current state of the environment. RL then determines the appropriate action based on the current state and evaluates the expected return of the action taken in the current state. The action and state spaces in traditional RL algorithms contain low dimensions and, therefore, are normally represented as tables or arrays for the approximate value functions. Real-world implementations, however, contain continuous and high-dimensional action and state spaces that cannot be represented as tables (curse of high dimensionality), requiring the approximate value functions to be represented as parameterised functions with weight vectors. This can be achieved using DRL, where the model features, policies and value functions are obtained using the function approximation and generalised to construct an approximation of the complete function through the deep neural networks. This, therefore, allows DRL to generalise states it didn't see during training and scale to large (potentially infinite) state spaces [49].

Deep Reinforcement Learning Algorithms

Deep Q-Network

Deep Q-Network (DQN) is a value-based DRL algorithm created by Mnih et al. [57] that combines Q-learning with deep neural networks to learn policies from high-dimensional sensory inputs. DQN contains a modified Q-learning implementation that uses an experience replay to store the agent's experiences in a replay memory at every time step. This replay memory is then accessed to execute weight updates [46]. These transitions are stored in memory as follows: $(s_t, a_t, s_{t+1}, r_{t+1})$. The RL agent then samples the transitions and trains from past experiences. An experience replay can reduce total interactions with the environment and variance by sampling batches of experience. To improve stability during training, the DQN uses a target network that updates its weights after a set number of steps to match the policy network's weights.

Using a fixed target network allows the DQN to avoid calculating the fluctuating TD errors stemming from its rapidly changing Q-value estimates [45]. The Double DQN algorithm, developed by Hasselt et al. [58], aims to reduce the overestimation problem in DQN when estimating the values of the actions. Double DQN does this by decoupling the action selection from the action evaluation (target network), improving value accuracy and policy quality.

Proximal Policy Optimisation

Proximal Policy Optimisation (PPO) is a policy gradient DRL algorithm that optimises the policy by alternating between collecting data from the policy through interactions with the environment and performing multiple optimisation steps using the collected data through stochastic gradient ascent. PPO achieves the same level of performance and efficiency of data as Trust Region Policy Optimisation (TRPO), a policy gradient algorithm which aims to optimise non-linear policies. Unlike TRPO, PPO uses only the first-order optimisation algorithm to achieve these results. When analysed on benchmark experiments such as playing Atari games and simulated robot locomotion, PPO performed better than current state-of-the-art approaches such as Advantage Actor-Critic (A2C) and performed similarly to Actor-Critic with Experience Replay (ACER) [59].

3 Literature Review

This chapter reviews existing literature on metaheuristic and RL algorithms used to solve different variants of the VRP. The first section of this chapter includes research that employs metaheuristic algorithms, such as Tabu Search, Genetic Algorithms and Simulated Annealing, whilst the second section includes research involving RL algorithms, such as DQN, A2C and REINFORCE.

3.1 Metaheuristic Algorithms for Vehicle Routing Problems

Li and Li [1] propose using an improved TS algorithm to solve the VRP with soft time windows and stochastic travel and service time (SVRP-STW). This variant of the VRP consists of a depot with a fleet of homogeneous vehicles that must find the optimal path to deliver goods to customers within a time window. In this problem, however, the vehicle can visit the customer outside the time window but will incur costs. The objective function consists of the penalty costs for arriving before or after the customer's time window and the total vehicle's travel cost. To solve the SVRP-STW, they propose using an improved TS algorithm. For the initial solution, containing the vehicles' routes, they implement a greedy algorithm that considers the capacity, time window and travel distance constraints when creating the initial routes. Throughout each iteration, customers close to the current node and with a limited time window are preferred and inserted into the current route at the most suitable feasible location. A new route is created when a vehicle reaches its capacity limit and cannot accept new customers. Their improved TS algorithm includes an adaptive tabu list that adjusts the tabu length interval based on the quality of solutions found and an adaptive neighbourhood structure that explores other neighbourhoods once a neighbourhood is explored. Compared to other metaheuristic algorithms, such as SA and GA, on 25, 50 and 100 customers, their proposed algorithm achieved impressive results in terms of execution time and solution quality, highlighting the effectiveness of this algorithm for solving SVRP-STW problems.

Gmira et al. [60] also propose using TS to solve a variant of VRP known as the Time-Dependent VRP with Time Windows on a Road Network $(TDVRPTW_{RN})$. This VRP variant aims to find the shortest path (in terms of time) for customers by considering the travel durations throughout the day. Similarly to Li and Li [1], they also generate the initial solution using the greedy algorithm; however, for the neighbourhood solution, they utilise the CROSS exchange algorithm [61] that swaps customers in a route without allowing reverse segments. This is suited for scenarios involving time-window constraints. To determine the feasibility of CROSS's exchange,

they use Dijkstra's algorithm to calculate the shortest distance between each pair of customers at a given departure time. In TS, they implement diversification to encourage the exploration of different regions within the solution space. They do this by employing a different objective over a set number of iterations, such as minimising the total distance rather than the duration. The original duration objective is then restored until the search process becomes trapped again in a local optimum. Their model successfully produced high-quality solutions in problem scenarios containing 200 nodes in 580 arcs within efficient execution times.

Rabbouch et al. [2] explored using GA to solve the Multi-Depot Heterogeneous VRP with Time Windows (MDHVRPTW). This problem involves finding minimal-distance routes for a heterogeneous fleet of vehicles situated in various depot locations to serve customers in different areas while adapting to their different time constraints and demands. They create the algorithm's initial population by assigning customers to the nearest depots and grouping them into clusters based on the number of depots. The algorithm encodes the chromosomes (solutions) as route sequences, encoding customers using their index in the order in which they will be visited. For the mutation and crossover, two parents are selected using tournament selection, which involves selecting the best two chromosomes depending on their fitness function. An offspring is created by randomly selecting a route from the first parent and replacing it with a route from the second parent. A customer is removed from the old route if it is already included in the new route. If a customer remains without a route, they are either inserted into the route in the appropriate position or added to a new route if it does not follow the time window constraints. Their proposed algorithm produced longer routes when compared to state-of-the-art algorithms, such as Variable Neighborhood Search (VNS) [62] and Unified TS (UTS) [63]. VNS is a metaheuristic algorithm that changes neighbourhood structures during a descent phase and an exploration phase to escape local optima [62] while UTS is an adapted TS that dynamically adjusts parameters to navigate the search space and applies diversification strategies to avoid local optima.[63]. The algorithm proposed by Rabbouch et al. [2], however, achieved impressive results in a shorter time, highlighting its effectiveness compared to VNS and UTS.

Park et al. [15] also explore using GA to solve the VRP while focusing on solving specifically the VRP with simultaneous pickup and delivery (VRPPD). This problem consists of a single depot with homogeneous vehicles that can concurrently pick up and deliver goods to customers using the same vehicle on a single route. In their proposed GA model, the VRPSPD solutions are encoded as chromosomes. Every gene contains a vehicle index value and links to three sets of parameter chromosomes: demand location, quantity, and product's weight variety index. The population consists of solutions, each representing a combination of randomly generated vehicle indices

assigned to nodes for delivery or pickup. The algorithm's fitness function included the total costs to deliver the products and the vehicle's total distance travelled based on the Euclidean distance equation. Selecting the parent chromosomes to create the child chromosomes involved using a roulette wheel selection. This consisted of calculating the fitness function for all solutions in population *S* and sorting them according to the highest fitness level. A number, *F*, is randomly selected between 0 and *S*. A partial sum, *P*, is then calculated using the cumulative sum of fitness values from the highest fitness on the list until *F*. The selected solutions in *P* contribute to creating the child's chromosomes. A crossover operator selects two random parents from the selected solutions and creates two child chromosomes by randomly creating a segment on each parent and replacing the segment from the second parent with the first parent segment. Their results indicated that the proposed model handled real-life scenarios within reasonable execution times.

Yu et al. [64] propose using SA to explore a variant of the VRP with Time Windows that includes the addition of delivering to parcel lockers. This variant is called the VRP with parcel lockers (VRPPL). In VRPPL, customers may have their items delivered to their house, nearest parcel locker, or either option. One vehicle can deliver once to a customer's house, whilst parcel lockers can be visited more than once by the same vehicle or different vehicles. The VRPPL's solution can be represented as a two-dimensional array of customer indices and delivery options. The objective function, which involves reducing the distance travelled, is calculated on the array's second row, consisting of the vehicle's visited nodes. For the initial solution, the customers and their delivery type are arranged in ascending order according to the customer's indices. The nearest neighbourhood algorithm then selects nodes closest to the currently visited node.

In their proposed model, the SA algorithm selects the initial solution as the current solution, $\sigma_{current}$, whilst the initial temperature T is set to T_0. The algorithm then uses the $\sigma_{current}$ to find a new solution, σ_{new}, by selecting one from the following neighbourhood moves on the first row: swap, insertion, and inversion. To determine which move is selected as σ_{new}, a random value r_1 is generated. The objective value for a solution σ is represented as $f(\sigma)$. σ_{new} is accepted as the new $\sigma_{current}$ by comparing $f(\sigma_{new})$ and $f(\sigma_{current})$, whilst σ_{new} is accepted as the new σ_{best} if $f(\sigma_{new}) < f(\sigma_{best})$. The difference between $\sigma_{new} - \sigma_{current}$ is represented as Δ. If σ_{new} has a higher objective value, it may still be accepted as a new $\sigma_{current}$ if $r_2 < e^{-(\Delta/\beta T)}$, where r_2 is a random value generated between 0 and 1, β is Boltzmann constant, $e^{-(\Delta/\beta T)}$ is the new neighbourhood solution probability. After an iteration is completed, a constant α is multiplied by T, reducing the temperature. Their findings showed that the SA performed better in small and large VRPPL instances than the Gurobi solver.

To solve the routing optimisation of ride-sharing taxis, Lin et al. [65] suggest

using SA to minimise costs and maximise customer satisfaction. The metrics used to evaluate these objectives included measuring the waiting time, extra riding time (time incurred due to ride-sharing) and travelling milage. In their proposed model, each vehicle travels at a constant speed, where the pick-up time windows, locations of the origin and destinations are known beforehand. Their results indicated that when serving 29 customers, the regular taxi system, which consisted of each taxi serving one customer at a time, would require 29 taxis and a total travel mileage of 375km to serve customers' demands. When repeating the same experiment using their SA model, only 10 taxis were used with a total travel miles of 302km. This indicated that their SA model saved 19% mileage and 66% of taxis remained available, indicating that fewer taxis were needed to serve customers.

Herbawi and Weber [66] also propose using metaheuristics to solve ride-sharing problems. Their research uses GA with an insertion heuristic to solve the ride-matching problem with time windows (RMPTW) in a dynamic ride-sharing environment. Similar to Lin et al. [65], their problem involves drivers and customers with origin and destination locations and a specified time window. Drivers may specify the total distance and travel time and whether they accept detours to accommodate customers. The RMPTW objective is to minimise the total distance and time of vehicles, minimise the time of customer trips, and maximise the total of customers served. The first stage of their model consists of using GA to solve the static rendition of the problem, which consists of the known requests. In the second stage, the model then uses the insertion heuristic to update the GA's solution to accommodate newly received requests. When demonstrated on real-world datasets, their model solved the dynamic ride-matching problem in real-time while providing solutions that balance quality and response time.

3.2 Reinforcement Learning for Vehicle Routing Problems

To improve ride-sharing services, Al-Abbasi et al. [67] present a model-free method that uses DQN to learn optimal dispatch policies that optimally assign vehicles to customers. Their objective was to minimise the supply and demand mismatch, customer waiting time, time users experience due to carpooling, and number of vehicles utilised, consequently reducing traffic congestion and fuel consumption. Each vehicle determines the best action using Q-learning by considering nearby vehicles and understanding how the action affects the reward. When evaluating their model on a New York City taxi dataset of 15 million taxi trips, their model outperformed baseline policies such as those not considering ride-sharing. Their model reduced the number of vehicles by 500 and the passengers' waiting time for the same number of requests.

Guo and Xu [68] explore solving the vehicle dispatching and routing problem in

a ride-sharing autonomous mobility-on-demand system using a Convolutional Neural Network and double DQN. By leveraging historical data, they utilize a DRL framework to make vehicle routing decisions that balance operation cost and service quality. Their model also considers the repositioning of idle vehicles where they relocated to regions with high demand. Using dynamic programming, their model then assigns vehicles to customers' requests based on vehicles that yield the highest reward. Their reward function consists of the passenger's quality of service subtracted from the system's operating cost. When experimented on a New York City dataset with 40, 60, and 80 identical vehicles, their model resulted in an increased service rate and a reduction in the vehicle's travelling distance and average waiting time when compared to other algorithms such as the optimal high-capacity vehicle dispatching algorithm with rebalancing proposed by Alonso-Mora et al. [69] and other strategies involving no rebalancing. Waiting time was, however, higher when compared to the strategy, which considered only the quality service of individual customers; nevertheless, their model compensated for this shortfall in terms of service rate and distance.

Vera et al. [3] propose utilising DRL to solve the Capacitated Multi-Vehicle Routing Problem (CMVRP), which consists of finding routes that minimise the total distance travelled by different vehicles (with different capacities) when serving customers' demands. Each agent's policy is represented through deep neural networks (DNNs) and is trained using RL. All agents (vehicles) access information from all other agents, ensuring a consistent environment state across all agents. They formulate the CMVRP as an MDP, where each agent makes decisions by relying solely on the information from the previous state of the environment and does not consider the actions of other agents. They use the A2C algorithm, a policy gradient method that uses the actor network and the critic network as function approximators to train the network. The stochastic policy is defined by parameters characterising its embedding, decoder, and attention mechanisms. The policy improves iteratively by estimating how changes in the policy parameters impact the expected rewards' gradient, thereby guiding the policy towards maximising expected rewards. The attention mechanism gathers information from all the elements in a set of input nodes, which it uses to create the output sequence. An affinity function is used to generate values that measure the relationship between each node and the previous output of the model. Applying the softmax function to these values provides the attention (importance) of each input element at each timestep, helping the model focus on relevant inputs when generating the output. Their proposed model performed better than the Sweep Heuristic [70], an algorithm that constructs routes from multiple depots and uses multiple heuristics to refine the solution, and the Clarke-Wright Savings Heuristic [71], an algorithm that finds efficient routes for multiple vehicles by combining pairs of delivery points into a single route to minimise total travel distance. However, when compared to Google

OR-Tools, it performed better. They stated that while OR-Tools performed better, their model could determine policies for any VRP configuration in faster execution times.

Similarly to Vera et al. [3], Zhao et al. [8] also propose using DRL to solve the VRP. Instead of applying a critic alongside an actor, Zhao et al. [8] use an adaptive critic. The actor is responsible for creating routing policies through an attention mechanism with graph embeddings, whilst the adaptive critic is a combination of self-critic and normal critic, responsible for optimising the actor's parameters during training to increase the convergence rate and improve performance. The adaptive critic calculates the policy gradient method's baseline to minimise variance during training. A routing simulator is used to create VRP instances that simulate real-world scenarios. It also acts as the model's environment, responsible for interacting with the model. After the vehicle chooses the next customer, the simulator provides the new states, rewards the actor and adaptive critic and updates customer masking. Customers who have already been visited or cannot be visited due to capacity or time constraints are masked. Their actor-network architecture consisted of the encoder (to embed the static and dynamic state to a D-dimensional space), the decoder (to decode the vehicle's embedding to a high dimensional state) and the attention layer (to learn the correlation between the customer point and the current state). Their results indicated that their DRL method achieved better results when compared to baseline algorithms such as ant colony optimisation (ACO). During their research, they also investigated using the DRL's output as the initial input for the local search algorithm using Google OR tools. They concluded that DRL could provide robust initial solutions for local search algorithms, leading it to find solutions that are close to optimal.

Basso et al. [7] use a modified Q-Learning algorithm to solve the Dynamic Stochastic Electric VRP (DS-EVRP). This problem involves determining routes for an EV serving random customers' requests throughout the day, starting and ending at a depot. Customer requests can be deterministic if acknowledged before departure or stochastic if received after the EV leaves the depot. The model uses a chance-constrained policy with two layers to learn a policy that minimises the vehicle's energy consumption and battery depletion when carrying out an action in a given state. They use value function approximation to estimate the value of state-action pairs using look-up tables with reduced state representation. They also use a training strategy that implements rollout heuristics to guide the agent in selecting the best actions when in a state. Their research shows that using RL to plan routes and anticipate charging, vehicles may save an average of 4.8% and up to 12% of energy.

To solve the VRP, Kool et al. [72] propose using an attention-based model consisting of an encoder and decoder. They propose training this model using the REINFORCE algorithm with a greedy baseline. Each attention layer in the model includes a multi-head attention and feed-forward sublayer. The encoder uses the

multi-head attention to embed the graph comprising the customer locations, whilst the decoder then uses these embeddings to output the next node.

Mustakhov et al. [5] propose using RL to solve the Stochastic Dynamic VRP (SDVRP). Using MDP to model the SDVRP, they implement the REINFORCE algorithm to learn an adjustable and responsive policy capable of adapting to the dynamic and stochastic demands of the problem. The optimal policy consists of maximizing the number of customers' demand. Contrary to Kool et al. [72], they use a dynamic encoder to encode the graph each time the agent selects an action. The decoder utilizes multi-head attention to decode the graph of the current state. Unlike Kool et al. [72], they take the current embedding of the graph, consisting of served and unserved customer demand at the current time and the vehicle's current location, as input to determine the next vehicle for the following action. They implement a baseline model that adjusts the parameters of the primary model. The baseline model calculates the reward using a greedy strategy that selects actions (nodes) that most likely yield the highest rewards based on their probabilities. On the other hand, the main model samples actions. The parameters of the baseline model are updated if there is a significant difference between the results of the two models.

Zhang et al. [73] propose using GAT with RL (GAT-RL) to solve the Multi-depot VRP with soft time windows (MD-VRPSTW). Their GAT-RL model uses an encoder with a multi-head attention mechanism to acquire and process location information from various depots and customers simultaneously. They integrate a GAT with the encoder to capture spatial and temporal details within the time window network, producing the outputs of the depots and customer embeddings. The decoder then uses the embeddings, the masking for constraints, and the vehicle's current state to generate sequences containing the customers to visit. In their research, they compare two policies: fixed-order policy and full-pair matching policy. The fixed-order policy uses a fixed order to assign vehicles to customers, whilst the full-pair matching policy expands the vehicle's exploration area. The model's parameters were trained using a policy gradient algorithm, REINFORCE, and a greedy rollout baseline. Their results indicated that the model with the fixed-order matching policy converged faster, although the model with the full-pair matching policy achieved better solutions. The GAT-RL model excelled in solution quality and efficiency compared to benchmark algorithms, such as hybrid GA with iterated local search.

Gupta et al. [74] use a combination of deep RL and heuristics to solve the Capacitated VRP with a time window (CVRPTW). They aim to reduce transportation costs by decreasing the number of vehicles in use and the total distance travelled. Similarly to Zhang et al. [73], their DQN model utilises a multi-head attention layer for the graph attention encoder. The model's decoder then takes the encoded state as input and utilises a fully connected network to generate Q-values for each action (each

customer). The Q-values help determine the action to select. Only feasible customer actions are considered while masking those not meeting the feasibility conditions. An insertion heuristic was used to improve the candidate solution obtained using deep RL. This heuristic involved splitting the routes with the least number of customers and then checking each node in these routes for feasibility by inserting them between the nodes in the remaining routes. The feasible node (location) with the least distance increase is selected and inserted to create a new route. Compared to GA, their model obtained close to optimal solutions with fewer vehicles deployed; however, it incurred an increase in distance travelled cost compared to GA.

Zhang et al. [4] also present a multi-head attention mechanism for an encoder-decoder architecture, which uses RL to develop a multi-agent attention model (MAAM). This model can solve the Multi-VRP with soft time windows (MVRPSTW). The encoder creates the depot and customers' embedding and updates these embeddings using multiple attention layers that perform multi-head attention and feed-forward operations. The decoder then uses as input the encoder outputs, the constraint mask matrix, and the context embedding to generate a sequence of customers for multiple vehicles, wherein, for each timestep, one customer is assigned to a vehicle. Like Zhao et al. [8], already visited customers and customers' demands that exceed the remaining vehicle's capacity are masked. Like [5] and [73], REINFORCE was used to train the MAAM with a baseline model that uses a greedy rollout policy to reduce variance and improve training efficiency during the training process. During each epoch, the policy parameters are updated if the difference between MAAM and the baseline model is significant. Their model outperformed classical heuristics such as GA and local search when experimented on 20, 50, 100 and 150 customers with different capacities. Their model provided robustness when handling different customer numbers and vehicle capacities, allowing it to be applicable in realistic scenarios.

Previous studies, such as [3], [8], [5], [73], [74] and [4], use an attention model (AM) to solve the VRP. In AM, the problem is represented as a graph. The AM uses neighbouring graphs to extract node features and create the solution incrementally using the decoder. The decoder uses predictions to generate a probability distribution across nodes, subsequently choosing one node to add to the partial solution based on this distribution. Peng et al. [75] state that the attention model's embeddings, consisting of the node features, remain fixed throughout and do not reflect the changes in the state as the model makes decisions. They explain that as the model creates a partial solution (a route consisting of a sequence of nodes that start and finish at the depot), the remaining unselected nodes can be considered a new instance that differs from the original instance. They, therefore, suggest that node feature embeddings should be updated based on this new instance. Figure 3.1 illustrates the partial solution generated by the model using the original instance and shows the new

instance created using the remaining nodes.

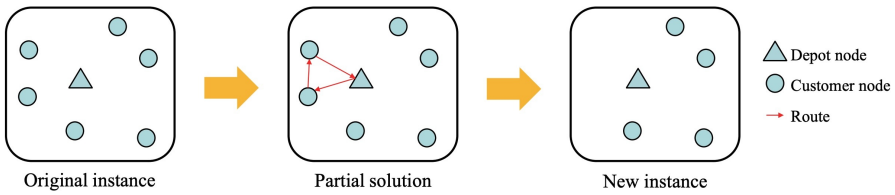


Figure 3.1 The different states of an instance at different steps (Extracted from: Peng et al. [75])

To address this problem, Peng et al. [75] present a dynamic attention model (AM-D) with a dynamic encoder-decoder architecture that updates each node's embeddings during different stages, such as whenever the vehicle returns to the depot. This change allows the model to dynamically discover node features and efficiently leverage hidden structural information across various steps. Unlike the vanilla encoder-decoder AM, the AM-D encoder-decoder updates each node's embeddings, including masking visited nodes when creating the partial solution. They train the AM-D using REINFORCE with a sample and greedy rollout. When experimented on instances with 20, 50 and 100 customers, the AM-D outperforms the AM across all problem instances. This indicates that the AM-D has strong generalization due to the smaller, easier-to-solve partial solutions created at each stage.

3.3 Summary

The research shows that metaheuristic algorithms can effectively produce high-quality solutions for different variants of the VRP. On the other hand, using RL algorithms to solve the VRP has also proven effective, with models able to learn policies for any VRP configuration. RL has also shown that it can serve as an excellent initial solution for local search algorithms. The RL models highlighted in the literature showed how using RL could optimise solutions more efficiently, leading to faster execution times when compared to traditional metaheuristic algorithms. RL's ability to be efficient and robust when handling instances with varying numbers of customers and vehicles highlights its ability to be applied in various real-world scenarios.

Recent research involving RL has incorporated an attention-based model to solve the VRP. This model utilises an encoder and decoder with a multi-head attention mechanism. Peng et al. [75] adapt this model to include a dynamic attention model with a dynamic encoder-decoder architecture, achieving a significant improvement in

terms of performance across different instances when compared to research, which uses the standard attention model to solve the VRP. This research can be further expanded and adapted to solve different variants of the VRP.

4 Methodology

As indicated in Chapter 3, most research focused on solving different variants of the VRP for logistic purposes, where they find the optimal routing strategies to enhance efficiency and reduce cost during transportation operations. Recent research highlights the use of RL together with an attention model. Peng et al. [75] expand upon this research to implement a dynamic attention model for the single depot VRP, which involves picking up items from a depot and delivering them to customers. In this chapter, we will combine the classic metaheuristics algorithms with machine learning techniques implemented by Peng et al. [75] to solve a variant of the VRP known as the MVRRPP. Unlike the problem investigated by Peng et al. [75], this problem adds complexity as it involves using multiple vehicles that must start and finish their route from their respective origin locations, which are different for each vehicle. For each customer, the vehicle must also visit the customer's respective pickup and drop-off location. This problem also requires the vehicles to visit specific locations sequentially, since a customer's pickup location must be visited before their respective dropoff location using the same vehicle.

4.1 Problem Formulation

The MVRRPP can be defined as a pair $X = \langle U, C \rangle$, where U represents a set of origin nodes $U = \{u_0, u_1, ..., u_{nu-1}\}$ with nu indicating the number of origin locations, whilst C represents a set of customer nodes $C = \{c_0, c_1, ..., c_{nc-1}\}$ with nc indicating the number of customers.

Each origin location $u_i \in U$ consists of a pair $\langle x_u, y_u \rangle$, where x_u is the latitude and y_u is the longitude of the origin coordinate. Each customer $c_i \in C$ consists of the pair $\langle p, q \rangle$, where p is the pickup location, q is the drop-off location. Each p consists of a pair $\langle x_p, y_p \rangle$, where x_p is the latitude and y_p is the longitude of the pick-up coordinate. Each q consists of a pair $\langle x_q, y_q \rangle$, where x_q is the latitude and y_q is the longitude of the drop-off coordinate. Each origin location u_i and customer c_i (pickup p_i and dropoff q_i) is two-dimensional and represented in Euclidean space.

A solution consists of a list of waypoints for each driver to visit. Each driver d_i consists of a list of waypoints $d_i = (w_0, w_1, ..., w_n)$, where each waypoint w represents a u_i or c_i (containing either pickup p_i or drop-off q_i). For each w_i containing a p_i , it's corresponding q_i must be visited by the same d_i. In each d_i , the sequence of w_i must first include the p_i before it's corresponding q_i . Each d_i has its respective origin location u_i and therefore must start and end from it. Since the number of drivers nd is equal to the number of origin locations, $nu = nd$.

The problem's final solution π , consisting of a list of drivers, can therefore be represented as $\pi = (d_0, d_1, ...d_{nd-1})$.

The current vehicle's occupancy (demand) of a d_i is defined as CS_i. For this work, it is assumed that the demand of a p_i always consists of one person, but we envisage that extending this for more persons should be relatively straightforward. With this assumption, the CS_i is incremented $CS_i + 1$ when visiting a p_i and reduced $CS_i - 1$ when visiting a q_i. Each d_i has a maximum seating capacity MS of 4. If $CS_i \geq MS$, then d_i can only visit a q_i until the CS is reduced. When $CS_i < MS$, d_i can continue picking up more customers.

4.1.1 Cost Function

The objective function of the MVRRPP involves minimising the total cost, which is the sum of the distance travelled by each vehicle, the customer's waiting time and travel time.

The distance between two waypoints $dist$ is calculated using the Euclidean distance formula:

$$dist(x, y) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{4.1}$$

where (x_1, x_2) and (y_1, y_2) are two w coordinates. By establishing $dist$, the total distance travelled by a driver $dist_d$ can be calculated as follows:

$$dist_d = \sum_{i=0}^{n} dist(w_{i-1}, w_i) \tag{4.2}$$

where the distances between the previous waypoint w_{i-1} and the current waypoint w_i is calculated for all the waypoints n in d_i (until the driver returns to the u_i). We consider the distance cost of the solution as the sum of distances travelled by all drivers:

$$Cost_{dist} = \sum_{d=0}^{nd-1} dist_d \tag{4.3}$$

The customer's waiting time and travel time are represented as a function of distance. This assumption is based on the fact that the further the driver's vehicle is from the customer, the more time it will take for the customer to be picked up. The distance can be converted to actual minutes by considering the vehicle's speed. The waiting time is the time spent by the customer waiting to be served by a vehicle (the time it took for the driver to arrive at the pick-up location p_i from the origin location u_i). Since time is a function of distance, the customer's waiting time is the cumulative

distance travelled by the vehicle up to the time of pickup. The cumulative distance CD_w of a driver d_i until a certain waypoint is calculated as follows:

$$CD_w(w_z) = \sum_{i=1}^{z} dist(w_{i-1}, w_i) \tag{4.4}$$

where the distances between each w_i in d_i is calculated until the index of the specified waypoint z is reached. By formulating CD_w, the customer's waiting time WT_c can be calculated as follows:

$$WT_c = CD_w(p_i) \tag{4.5}$$

where the CD_w travelled to a customer's pickup location p_i in d_i is calculated. We then calculate the driver's total waiting time DWT_d for all customers in d as follows:

$$DWT_d = \sum_{c=0}^{nt-1} {WT_c}^2 \tag{4.6}$$

where the waiting time for each customer in d is calculated and squared until the number of customers in d, represented as nt, is reached. The waiting time is squared so that it is distributed equally between all the customers. By squaring, we are amplifying the waiting time cost of each customer; therefore, solutions that keep the waiting time low across each customer are preferred compared to solutions containing a mixture of low or high waiting times, as the high ones are amplified by the squaring, increasing the overall waiting time cost. The sum of the total waiting time cost of all drivers $Cost_{wt}$ is then calculated using the following formulation:

$$Cost_{wt} = \sum_{d=0}^{nd-1} DWT_d \tag{4.7}$$

Another objective function to consider is the travel time, the time a driver takes from a customer's pickup to the customer's drop-off (the time a customer spends inside a vehicle). The customer's travel time TT_c can be calculated as follows:

$$TT_c = CD_w(q_i) - CD_w(p_i) \tag{4.8}$$

where the customer's waiting time $CD_w(p_i)$ is subtracted from the customer's dropoff time $CD_w(q_i)$. The driver's total travel time DTT_d of all customers in d can then be calculated using:

$$DTT_d = \sum_{c=0}^{nt-1} {TT_c}^2 \tag{4.9}$$

where the travel time for each customer in d is calculated and squared until nt is reached. Similarly to the waiting time, the travel time is also squared for the same reason; to prefer solutions that distribute the travel time equally between customers. The total travel time of all drivers $Cost_{tt}$ can then be calculated as follows:

$$Cost_{tt} = \sum_{d=0}^{nd-1} DTT_d \tag{4.10}$$

The cost function's overall cost $Cost_{total}$ can therefore be calculated as follows:

$$Cost_{total} = Cost_{dist} + Cost_{wt} + Cost_{tt} \tag{4.11}$$

For evaluation purposes, we then calculate the travel impact time for each customer. This is the additional time a customer incurs due to ride-pooling, the time after pickup, where the driver may pick up additional customers before finally dropping off the customer. We first determine the optimal travel time. This is time it takes for a vehicle to travel directly from the customer's pickup location to the destination without ride-pooling. The optimal travel time for a customer OT_c is calculated as follows:

$$OT_c = dist(p_i, q_i) \tag{4.12}$$

where the distance between the customer's pickup p_i and drop-off q_i is calculated. By obtaining the OT_c, we can then determine the travel impact time IT_c of the customer using the following formula:

$$IT_c = TT_c - OT_c \tag{4.13}$$

where the optimal travel time of the customer OT_c is subtracted from the actual travel time TT_c of the same customer .

4.2 The Dataset of Problem

Peng et al. [75] generate a VRP instance by creating a set of nodes representing the depot and customers' locations. Each node contains a randomly generated 2-dimension coordinate in Euclidean space from the unit square of [0,1] x [0,1] and a randomly generated demand containing a discrete value between 1 and 9, except for the depot, which has a demand of 0.

The dataset for this project was generated similarly to that of Peng et al. [75] but adapted to include the following: multiple depots (where for the MVRRPP, we consider the depots as the origin locations), the removal of demand (since it's assumed that the demand of each pickup is always one) and the modification of the customers'

nodes (where for each customer there is a pickup and dropoff location). This dataset was generated by creating three tensors for the origin, pickup and drop-off locations using TensorFlow, an open-source library for machine learning. The origin tensor represents the drivers' starting and finishing location of a route. Each tensor has a three-dimensional shape consisting of the number of samples, the number of drivers/customers and 2D coordinates. The first dimension represents the number of samples, each representing a different MVRRPP instance. The second dimension specifies the number of drivers for the origin tensor and the number of customers for the pickup and drop-off locations since the number of pickups and drop-offs equals the number of customers. The third dimension represents a randomly generated two-dimensional coordinate (x,y) consisting of the latitude and longitude. The latitude and longitude values are randomly generated integers between 0 and 1. The three tensors are then combined to create a dataset consisting of all the coordinates needed for the MVRRPP. Different datasets were created to evaluate the MVRRPP on various scenarios containing a number of different customers and drivers. For each group of 4, 5, and 6 drivers, a different dataset was created with 20, 50, and 100 customers. Each dataset consisted of 500 samples (different MVRRPP instances). Figure 4.7 illustrates the dataset in the tensor format.

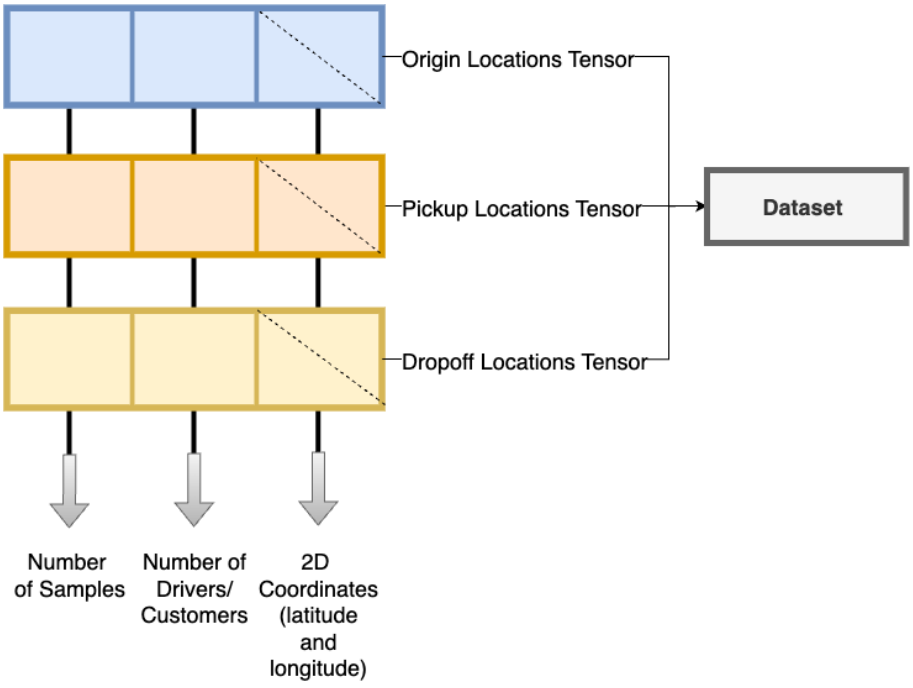


Figure 4.1 Diagram illustrating the dataset of the problem

4.3 Objective 1: Using Tabu Search as a baseline

As highlighted in Chapter 3, metaheuristic algorithms are frequently used to address different variants of the VRP. For the first objective of this research, a TS metaheuristic algorithm was implemented as a baseline to solve the MVRRPP. The performance of the implemented TS was then evaluated in terms of cost and computational times.

4.3.1 Formulating the Initial Solution

An initial solution was constructed for the TS, which involved creating the initial routes for each driver. Each route is a sequence of locations visited by a driver. To create the initial solution containing the initial routes, the customers were distributed equally to each driver's route in a round-robin manner. When assigning a customer to a driver's route, their pickup is assigned first, followed by their drop-off locations. After assigning all customers to the drivers' routes, each driver's origin location is added to the start and end of their respective routes. This indicates that each driver must start their journey from their origin location and finish at that same origin location.

Figure 4.2 illustrates the formulation of the initial solution, where for each customer in the instance, their pickup (represented by an orange circle) and dropoff (represented by a yellow circle) location were assigned to a driver. This process was repeated until all customers were distributed equally among drivers. Each driver's origin location (represented by the blue circles) was then assigned to the beginning and end of their respective routes. The diagram shows how all drivers' routes form the initial solution.

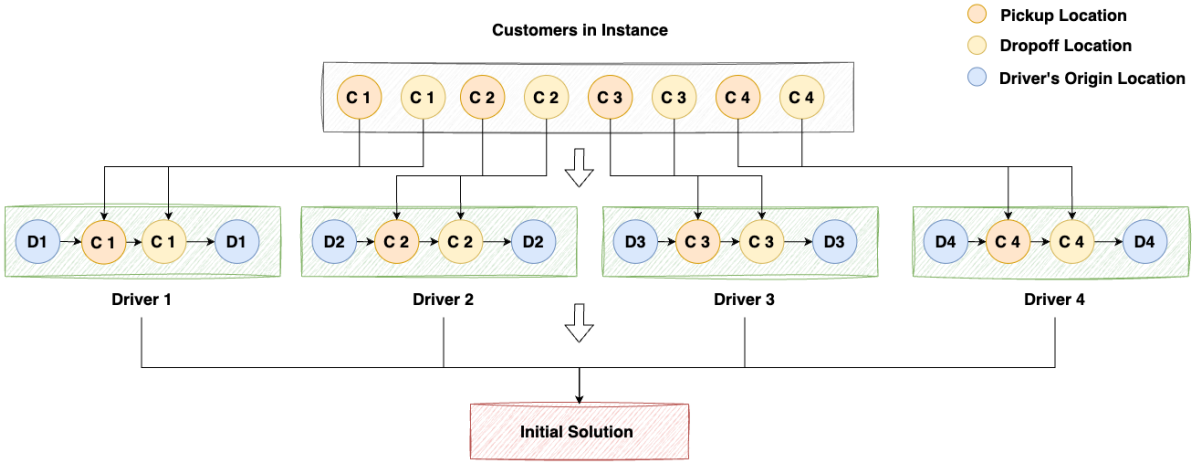


Figure 4.2 Diagram illustrating the formulation of the initial solution

4.3.2 Creating the Neighbourhood solutions

In TS, the neighbourhood solutions are a set of new solutions obtained from the current solution when modified using certain moves. The neighbourhood solutions allow the algorithm to explore the solution space and find better solutions from the current one. As previously mentioned, the solution consists of a list of driver routes, each starting and finishing at their respective origin location. Two types of moves were applied when creating the neighbourhood solutions. These moves involved swapping the pickup and drop-off nodes within the same driver's route and moving pickup and drop-off nodes to the beginning or the end of another driver's route.

The first move consisted of looping through each driver's route and swapping each route's location with every other location in that route, excluding the first and last location (origin location). After swapping a location, the new route's sequence is verified to check whether each customer's pickup location is before its respective drop-off location. If the drop-off location is before the pickup after a swap, that solution is omitted, and the algorithm proceeds with the next swap. The vehicle's occupancy is also checked after each swap to ensure its capacity has not been exceeded throughout the sequence. This involved counting the number of customers in a vehicle after a pickup. If the number of customers exceeds 4, that solution is omitted. If, after a swap, the solution satisfies each of these constraints, the newly updated route and the remaining drivers' routes are added to the list of neighbourhood solutions as one solution.

Figure 4.3 illustrates the implementation of the swap moves, where the blue circles represent the driver's origin location, the orange circles represent the customer's pickup, and the yellow circles represent the customer's dropoff. The diagram shows an example of the driver's one route and two solutions demonstrating the horizontal swap moves on this route. In the first solution, swapping the 'C1' dropoff location with the 'C2' pickup will result in a valid solution since both 'C1' and 'C2' pickups would remain before their respective dropoff locations. This solution is therefore added to the neighbourhood solutions. In the second solution, swapping the 'C1' dropoff location with the 'C2' dropoff location would result in an invalid solution since the 'C2' dropoff will now appear in the route's sequence before its corresponding pickup location.

The vertical moves involve looping through each driver's route and inserting each customer's pickup and drop-off location of that route into the beginning and end of the remaining drivers' routes. When inserting a customer's pickup and drop-off location into a new driver's route, they are removed from their original route and inserted into another driver's route. The new and remaining routes are then added as one solution to the list of neighbourhood solutions. This process is repeated until the customer's locations are inserted into each driver's route. Each insertion consists of

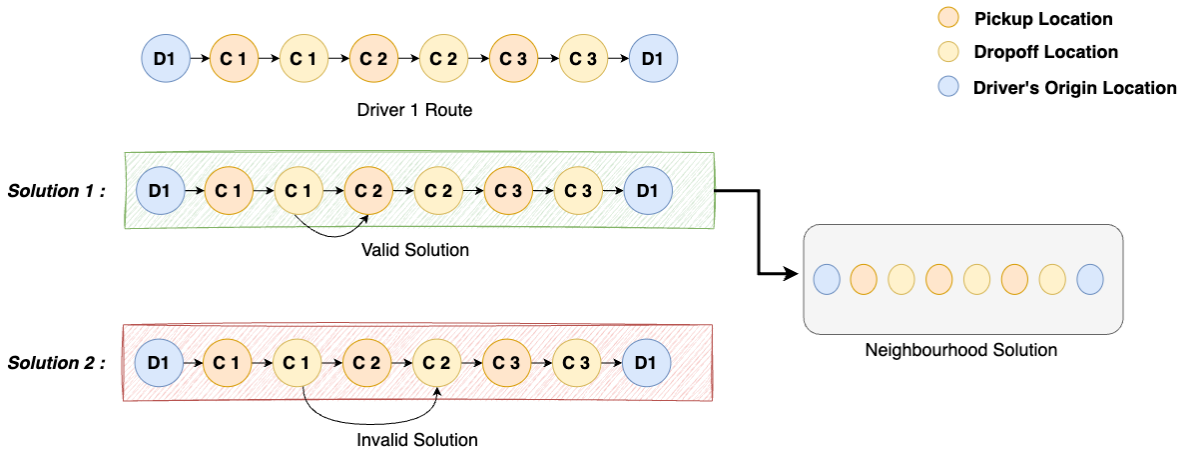


Figure 4.3 Diagram illustrating the horizontal swaps of the neighbourhood solution

adding the locations to the start (after the first location) and end of the route (before the last location), with each insertion considered as a separate solution.

The vertical moves can be formulated as follows: for the same customer c_i, the waypoint w_i consisting of the customer's pickup p_c and the waypoint w_j consisting of the customer's dropoff q_c are removed from a driver's solution d_i . The pickup-dropoff waypoints are then either:

- prepended to another driver solution d_j : d'_j = (w_i, w_j) $\oplus$ d_j
- appended to another driver solution d_j : d'_j = d_j $\oplus$ (w_i, w_j)

where $\oplus$ is the list concatenation operation, and d'_j is the new solution for driver j after the move. Since the pickup-dropoff pair is moved to the new route in the correct order (pickup before dropoff) and the driver always starts and ends the route with no occupancy, no constraints are violated after this move.

Figure 4.4 shows an example of the neighbourhood solution's vertical moves. This diagram contains the routes of two drivers: Driver 1 and Driver 2. The first customer's pickup location (orange circle) and dropoff location (yellow circle) in Driver 1's route are moved to the start of Driver 2's route after the origin location (blue circle). The new routes of Drivers 1 and 2 are then added as one solution to the neighbourhood solution.

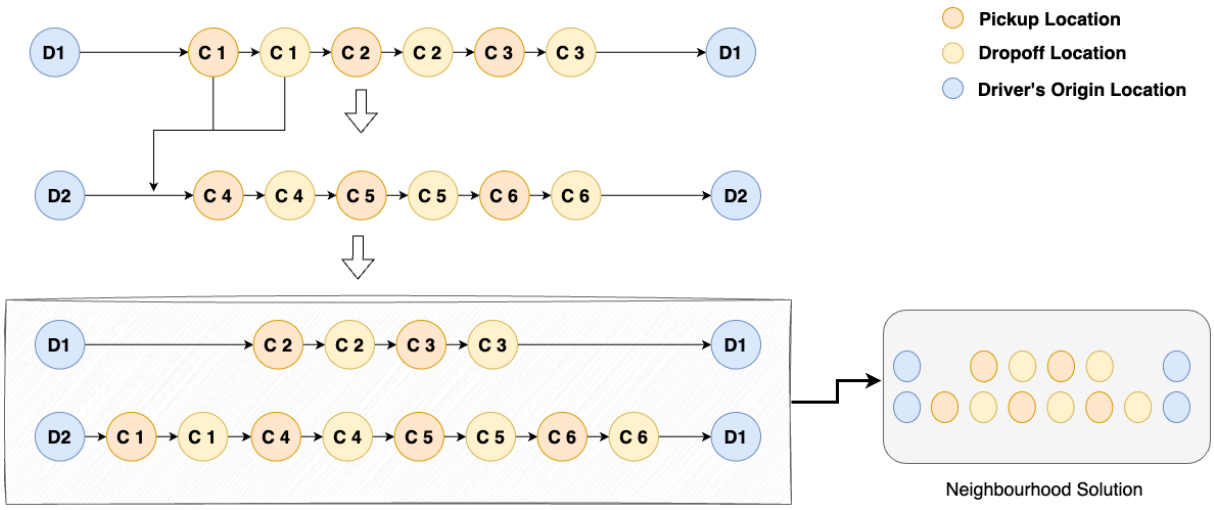


Figure 4.4 Diagram illustrating the vertical moves of the neighbourhood solution

4.3.3 Finding the Best Candidate

At the start of the algorithm, TS generates the initial solution containing each driver's route and adds it to the tabu list. Given the initial route, a set of different neighbourhood solutions are generated using the moves specified in Section 4.3.2. Each solution in the neighbourhood solution is known as a candidate and is evaluated using the cost function defined in Section 4.1.1. If a candidate has a lower cost than the current 'best candidate' and wasn't already added to the tabu list, that candidate becomes the new 'best candidate'. This new 'best candidate' is then added to the tabu list. If the tabu list exceeds its predefined size of 10 candidates, the oldest candidate will be removed. The 'best candidate' cost is then compared with the 'best solution' cost; if lower, the 'best candidate' becomes the new 'best solution'. The process is repeated for 100 iterations, where for each iteration, the current 'best candidate' of the previous iteration is used to generate new neighbourhood solutions. A stopping criteria was implemented that stops the TS from executing further iterations if the 'best solution' has not improved after 20 iterations. After executing the number of iterations, the algorithm will return the best solution found during the TS process.

The performance of TS was evaluated on multiple datasets, each including different scenarios with a different number of drivers and customers, as highlighted in Section 4.2. For each dataset, the TS algorithm was executed on 500 different instances (samples). The performance of the TS on a dataset (problem scenario) was assessed by taking the average performance of each metric on the 'best solutions' obtained across the 500 instances. This involved evaluating the average total distance, waiting time, and travel impact time. In addition, the actual time taken (in seconds) to solve each instance in the dataset was also recorded, along with the number of iterations needed to find the 'best solution'.

4.4 Objective 2: Find the appropriate representation of states, actions and reward to model as a RL Problem

For the second objective of this research, an RL algorithm using a dynamic attention model was implemented. As previously mentioned, the design of the attention model was based on research by Peng et al. [75]. We extend and adapt this work to solve the MVRRPP.

4.4.1 State

The state contains the information describing the current environment for the agent to be able to decide on the next action. The initial state encodes the coordinates of the origin, pickup and drop-off locations. Table 4.1 shows the information recorded in the state.

Table 4.1 Information recorded in the state

Observation	**Data Stored**	**Data Type**	**Dimensions**
Available Pickups	0 - Not visited , 1 - Visited	Binary	(batch size, number of customers)
Available Drop-offs	0 - Not visited , 1 - Visited	Binary	(batch size, number of customers)
Current driver's origin location	Index of origin location in action space	Int	(batch size, 1)
Number of occupied seats in the driver's vehicle	Number of customers picked up but not yet dropped off	Int	(batch size, 1)
Driver's Distance Travelled	Total distance travelled in a route	Float	(batch size, 1)
Customer's Waiting Time	The waiting time of each customer in a route	Float	(batch size, number of customers)
Customer's Travel Time	The travel time of each customer in a route	Float	(batch size, number of customers)

The state records the available customers' pickup and drop-off locations. Visited locations are updated using binary values where a value of '1' indicates that the location has been visited, while a value of '0' indicates that it has not been visited yet. Keeping a record of the visited locations allows us to create a mask (explained further in the decoder subsection) that prevents the agent from revisiting locations. The state also keeps track of the current driver's origin location, which is also used for masking

purposes to prevent the agent from selecting a different origin location when returning to the starting location. The state stores the number of occupied seats of the current driver's vehicle to prevent the agent from selecting further customers while the vehicle is full. The state also stores each driver's travelled distance, customer's waiting time, and travel time to calculate the reward function (explained further in Section 4.4.3), which is used to determine the performance and quality of the solution.

Following a similar approach to Peng et al. [75], a dynamic encoder-decoder architecture was implemented where, during different steps, the feature embeddings of the nodes are dynamically updated based on the current instance. The following subsections include a detailed explanation of the dynamic encoder-decoder architecture of the model. Figure 4.5 illustrates the data flow in the encoder-decoder architecture, starting from the graph instance.

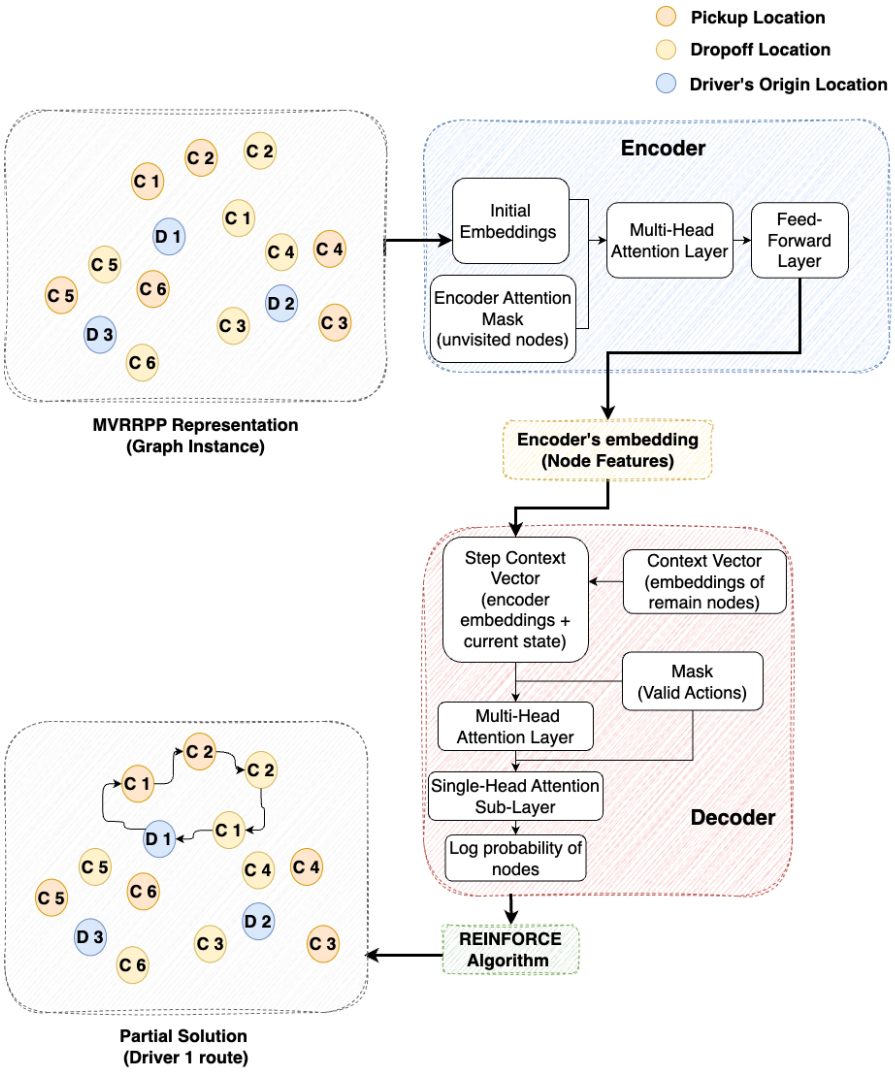


Figure 4.5 Diagram illustrating the data flow of the encoder-decoder architecture

State Encoder

An encoder was implemented using a GAT, whose purpose is to encode the information of the MVRRPP representation to an embedding. The MVRRPP representation consists of the nodes' coordinates, organised as a tuple of 3 tensors, each containing the waypoint w_i coordinates for the origin points U, pickups P and drop-off locations Q. Each waypoint node w_i has dimension of $d_x = 2$. For each tensor, the encoder uses a dense layer in the GAT that takes the input data of the tensor and forms the initial embeddings for that tensor.

By carrying out hyperparameter optimisation before training (explained in more detail in Section 4.4.5), the optimal set of hyperparameters that minimised the cost function was found. One of these hyperparameters included the optimal embedding dimension size d_h. The dense layers use this optimal embedding dimension size d_h to specify the number of output units for the initial embedding (the output space dimensionality). The dense layers apply linear transformations using learnable parameters for the weights $W \in \mathbb{R}^{d_h \times d_x}$ and bias $b \in \mathbb{R}^{d_h}$. Each tensor has it's own parameters: W_u and b_u for the origin coordinates, W_p and b_p for the pickup coordinates and W_q and b_q for the drop-off coordinates. During training, the W and b of these layers are adjusted to capture patterns in the input data that are beneficial for solving the MVRRPP. The initial node embedding $h_i^{(0)}$ can be represented using the following function:

$$h_i^{(0)} = \begin{cases} W_U x_i + b_U & \text{if } i \in U \\ W_P x_i + b_P & \text{if } i \in P \\ W_Q x_i + b_Q & \text{if } i \in Q \end{cases} \tag{4.14}$$

Each embedding is concatenated to form the initial embeddings of the graph. The initial embeddings and an attention mask are then passed through three stacked attention layers $S = 3$, each containing a multi-head attention (MHA) sublayer and a fully connected feed-forward (FF) sublayer. The attention layers aim to capture the relationships between nodes using an attention mechanism that assigns the attention weights to different parts of the input sequence. In the attention layer $\ell \in \{1, ..., S\}$, each layer's node embedding is represented as h_i^ℓ, where each node is i. The output of layer $\ell - 1$ is $\left\{h_0^{(\ell-1)}, ..., h_s^{(\ell-1)}\right\}$ which also represents the input of ℓ.

The attention mask guides the multi-head attention mechanism to focus on unvisited nodes and exclude nodes that were already visited in the current partial solution $\pi_{1:t-1}$, where $t \in \{1, ..., T\}$ represents each step. It dictates which nodes the attention mechanism can attend to in the GAT. An attention mask (represented as a binary matrix) was created, where the 0s indicate locations that should be considered (yet to be visited), and 1s indicate locations that should be ignored (already visited).

The attention mask is updated whenever a driver returns to their origin location to reflect the current state of the visited origin, pickups, and drop-off locations.

Each multi-head attention sub-layer uses 8 attention heads $M = 8$, where each attention head is represented as $m \in \{1, ...M\}$. The initial embedding tensor is used for the queries $r_{im}^{(\ell)} \in \mathbb{R}^{d_k}$, keys $k_{im}^{(\ell)} \in \mathbb{R}^{d_k}$, and values $v_{im}^{(\ell)} \in \mathbb{R}^{d_v}$ tensor. They are projected into several subspaces using linear transformations by multipling each tensor with their respective weight matrices $W_m^R \in \mathbb{R}^{d_k \times d_h}$, $W_m^K \in \mathbb{R}^{d_k \times d_h}$ and $W_m^V \in \mathbb{R}^{d_v \times d_h}$, where d_k = d_v = $\frac{d_h}{M}$, with d_k representing the dimensions of key and d_v representing the dimensions of value. These transformed tensors are then split into multiple heads to execute the attention computations simultaneously across each head in parallel. The r, k, and v vectors for each attention head m in the multi-head attention layer ℓ, can be represented as follows:

$$r_{im}^{(\ell)} = W_m^R h_i^{(\ell-1)}, k_{im}^{(\ell)} = W_m^K h_i^{(\ell-1)}, v_{im}^{(\ell)} = W_m^V h_i^{(\ell-1)} \tag{4.15}$$

The compatibility scores between the r and k matrices are computed using the matrix multiplication operation. The compatibility scores capture the similarity between the r and k, which are required for determining the attention weights. These compatibility scores are then rescaled by dividing the scores with the square root of the dimension of k ($\sqrt{d_k}$) to control the magnitude of the compatibility matrix. The attention mask specified earlier is then applied to the compatibility scores, by assigning a negative infinity value to visited nodes and therefore preventing attention from being assigned to masked positions (visited nodes). The compatibility scores can be formulated as follows:

$$u_{ijm}^{\ell} = \begin{cases} \left(r_{im}^{(\ell)}\right)^T k_{jm}^{(\ell)} & \text{if } x_j \notin \pi_{1:t-1} \text{ or } j \in U \\ -\infty & \text{otherwise} \end{cases} \tag{4.16}$$

where if node x_j is not in the partial solution $\pi_{1:t-1}$ or node j is an origin node in the set of origin points U, the compatibility scores are calculated, otherwise an negative infinity value is assigned.

The softmax function is applied to the compatibility scores to obtain the normalized attention weights for each element of the input sequence. This can be formulated using the following formula:

$$a_{ijm}^{(\ell)} = \frac{e^{u_{ijm}^{(\ell)}}}{\sum_{j'=0}^{n} e^{u_{ij'm}^{(\ell)}}} \tag{4.17}$$

where the softmax function is calculated by finding the exponential of the attention score $e^{u_{ijm}^{(\ell)}}$ and dividing it over the sum of the exponential of the attention

scores of all nodes j', defined as $\sum_{j'=0}^{n} e^{u_{ij'm}^{(\ell)}}$.

The attention weights are multiplied by $v_{im}^{(\ell)}$ to calculate the weighted sum of v for each position in the r sequence. These attention weight allow the attention mechanism to attend to specific parts of the input data and capture the relevant information from the input sequence. The weighted sum of v can be represented as follows:

$$h_{im}^{'(\ell)} = \sum_{j=0}^{n} a_{ijm}^{(\ell)} v_{jm}^{(\ell)} \tag{4.18}$$

For each m, the weighted sum of v, $h_{im}^{'(\ell)}$ is passed through a dense layer, where linear transformation is applied using the weight of the respective m head, $W_m^Y \in \mathbb{R}^{d_h \times d_v}$. The sum of all m heads outputs represents the output of the multi-head attention layer. The output of the multi-head attention layer for node i at layer ℓ can therefore be represented as:

$$\mathsf{MHA}_i^{(\ell)}\left(h_0^{(\ell-1)}, ..., h_n^{(\ell-1)}\right) = \sum_{m=1}^{M} W_m^Y h_{im}^{'(\ell)} \tag{4.19}$$

The initial embeddings and the multi-head attention layer output are combined using a skip connection. The purpose of the skip connection is to mitigate the vanishing gradient problem during training as gradients backpropagate through multiple layers. The skip connection allows the gradients to flow directly to subsequent layers, helping it to preserve information from the original input (the input embeddings).

The output of the skip connection is then passed through a hyperbolic tangent activation function (tanh), where non-linearity and normalization (within the range of -1 to 1) are applied. This can be represented using the following formula:

$$\hat{h}_i^{(\ell)} = \mathsf{tanh}\left(h_i^{(\ell-1)} + \mathsf{MHA}_i^{(\ell)}\left(h_0^{(\ell-1)}, ..., h_n^{(\ell-1)}\right)\right) \tag{4.20}$$

After the tanh activation, the output is processed through two feed-forward layers. Each feed-forward layer applies a linear transformation followed by an activation function, each using different trainable parameters: $W_0^F \in \mathbb{R}^{d_F \times d_h}$ and $b_0^F \in \mathbb{R}^{d_F}$ for the first feed-forward layer and $W_0^F \in \mathbb{R}^{d_F \times d_h}$ and $b_1^F \in \mathbb{R}^{d_F}$ for the second feed-forward layer, where $d_F = 4 \times d_h$. The rectified linear unit (ReLU) activation is applied after the first feed-forward layer, which applies non-linearity, removing any negative values and replacing them with zeros. This is outlined through the formula presented below:

$$\mathsf{FF}\left(\hat{h}_i^{(\ell)}\right) = W_1^F \mathsf{ReLu}\left(W_0^F \hat{h}_i^{(\ell)} + b_0^F\right) + b_1^F \tag{4.21}$$

The tanh activation is then applied after the second feed-forward layer, which is the final output of the attention layer. The output of a node after passing through the multi-head attention layer and the feed-forward sublayer can be represented as follows:

$$h_i^{(\ell)} = \text{tanh}\left(\hat{h}_i^{(\ell)} + \text{FF}\left(\hat{h}_i^{(\ell)}\right)\right) \tag{4.22}$$

The output of **S** stacked attention layers represents the output of the encoder. The final node embedding h_i^S for node i after S attention layers is formulated as follows:

$$h_i^S = \text{ENCODE}_i^S\left(h_0^0, ..., h_s^0\right) \tag{4.23}$$

Decoder

A decoder was implemented to generate the solution sequence for the MVRRPP. The decoder is responsible for selecting a node at each step. Similarly to the encoder, the decoder used multi-head attention to allow the model to learn complex relationships and capture higher-level features in the context of the given decoding step. A step context vector was created using the embeddings generated by the encoder and the current environment's state. This vector captures information about the MVRRPP's current state, which is essential for making decisions in the subsequent steps of the decoder stage.

The step context vector consists of the previous node's embeddings $h^S_{\pi_{t-1}}$ if at step $t-1$ or the first chosen origin point embedding h_O^S if at step $t=1$, the remaining seating capacity of the current vehicle RS_t, the total distance travelled of all vehicles TD_t, the number of drivers left TO_t, the number of unpicked customers TC_t, the indexes of the remaining drivers RD_t and the indexes of the remaining customers' pickup RP_t and dropoff RQ_t. The step context vector $h_c^{'}$ is formulated using the following formula:

$$h_c^{'} = \begin{cases} \left[\bar{h}_t; h_O^S; RS_t; TD_t; TO_t; TC_t; RD_t; RP_t; RQ_t;\right] & \text{if } t = 1 \\ \left[\bar{h}_t; h^S_{\pi_{t-1}}; RS_t; TD_t; TO_t; TC_t; RD_t; RP_t; RQ_t;\right] & \text{if } t > 1 \end{cases} \tag{4.24}$$

where $\bar{h}_t$ represents the mean graph embeddings of unvisited nodes and $;$ represents the concatenation in the step context. The step context vector is passed through a dense layer, which applies a linear transformation using a learned weight matrix at each decoding step. The output of this layer projects the step context vector into a new space compatible with the decoder's attention mechanism. This projected step context vector is then added to the context vector $Q\_context$.

The $Q\_context$ consists of the mean embedding for the remaining nodes. It is

achieved by dividing the encoder's embeddings with the number of remaining available nodes (origin, pickup, and dropoffs) and passing it through a dense layer for linear transformation. Adding the context vector with the step context vector results in the final query tensor r for the decoder's multi-head attention. This r vector provides context information on the remaining unvisited nodes and the agent's current state. The r vector was split into M attention heads to allow the parallel processing of the multi-head attention computations, where each head attends to different aspects of the query vector.

The decoder attention mechanism's key k and value v projections were calculated by passing the encoder's embedding h_j^S through dense layers allocated for k and v, and applying a linear transformation with the weight matrix W^K and W^V respectively. These parameters are different from the ones used in the encoder. The k and v tensors are each split into M heads for parallel processing. The decoder's r, k and v can be formulated using the following formula:

$$r\left(c\right)_m = W_m^R h_c^{'},\ k_{jm} = W_m^K h_j^S,\ v_{jm} = W_m^V h_j^S \tag{4.25}$$

where $r\left(c\right)_m$ represents the query vector for the context vector $h_c^{'}$ in head m and k_{jm} and v_{jm} represent the key and value respectively for the current node j in head m.

A mask was created to guide the decoder's attention mechanism. The mask ensures the agent selects a valid node as its next action by excluding (masking out) invalid actions that do not adhere to the rules and constraints of the MVRRPP. Upon selecting the initial action, the mask excludes all nodes except the origin locations, allowing the agent to select an origin point to start the route. Once the agent selects an origin location, all origin locations are masked out, and the available pickup locations are unmasked. The remaining origin locations are masked until the agent returns to the selected origin location. Once the agent selects a customer's pickup location, that pickup location is masked, and the respective customer's drop-off location is unmasked. At this stage, the mask allows the agent to select another pickup location or drop off the picked-up customer.

When an agent visits a pickup location, the occupancy is incremented by 1. If the vehicle's occupancy reaches capacity (which was set to 4), all available pickup locations are masked out to prevent the agent from picking up further customers. If a customer is dropped off, the occupancy is decreased by one, and the available pickup locations are again unmasked since the vehicle can now pick up more customers. If, at a particular step, there are no drop-off locations left because the customers picked up have been all dropped off or due to all customers in the instance having been dropped off, then the selected origin location becomes unmasked, allowing the agent to return to the origin location.

When an agent returns to the selected origin location, that origin location becomes masked along with the pickup locations (if there are any left). The remaining origin locations are then unmasked, allowing the agent to select another origin location for the next route. This process is repeated until all customers have been dropped off or until there are no further origin locations from which to start a new route. The mask is of a boolean type, where valid nodes have 'False' values whilst invalid nodes have 'True' values. The decoder's mask DM can be represented as a list of boolean values dm_i, where the size of the list is equivalent to the total number of nodes in the instance, tn (number of drivers nd + number of pickup and dropoffs $2(nc)$). The DM can be formulated as follows:

$$DM = (dm_1, dm_2, ...dm_{tn}) \tag{4.26}$$

The decoder's multi-head attention is computed using its r, k, and v vectors and mask DM. Similarly to the encoder, the compatibility scores are calculated by finding the matrix multiplication between r and v, followed by dividing the square root of the dimension of the head depth. The decoder's mask is then applied to the compatibility scores, where the elements equivalent to a 'True' value (indicating a masked node) are assigned a negative infinity value. The compatibility score between the transposed query vector $r^T_{(c)m}$ and the key k_{jm} of node j in head m can be formulated as follows:

$$r_{(c)jm} = \begin{cases} r^T_{(c)m} k_{jm} & \text{if } dm_j = False \\ -\infty & \text{otherwise} \end{cases} \tag{4.27}$$

where the compatibility score is calculated if dm_j, representing the index of node j in the decoder's mask DM is 'False' otherwise a negative infinity value is assigned.

The attention weight of each element $a_{(c)jm}$ in the input sequence was then calculated by applying a softmax function to the masked compatibility scores. By applying the softmax function, the negative infinity values in the masked compatibility score resulted in a zero probability (weight) and, therefore, will not influence the attention distribution. The attention weight for node j can be formulated using the following formula:

$$a_{(c)jm} = \frac{e^{u_{(c)jm}}}{\sum_{j'=0}^{tn} e^{u_{(c)j'm}}} \tag{4.28}$$

The weighted sum of the v vectors for each element in query r was then determined by applying a matrix multiplication between the attention weights $a_{(c)jm}$ and the v_{jm} vectors. These attention scores are then reshaped to include information from all attention heads. The weighted sum $h'_{(c)m}$ can be formulated as follows:

$$h'_{(c)m} = \sum_{j=0}^{tn} a_{(c)jm}\, v_{jm} \tag{4.29}$$

The final output of the decoder's multi-head attention is calculated by applying a linear transformation using its learnable weight W_m^Y . This output contains the context information for the current decoding step. The output of the decoder can be presented using the below formula:

$$h_c = \sum_{m=1}^{M} W_m^Y\, h'_{(c)m} \tag{4.30}$$

The next stage of the decoder is to calculate the log probability when selecting each node in the MVRRPP. The log probability is determined using a single-head attention sub-layer with the following inputs: the multi-head attention mechanism's output (represented as $r = W^R h_c$), the projection of node embeddings (represented as $k_j = W^K h_j^S$) and the decoder's mask DM. The compatibility scores were first calculated using the dot product between the r and k vectors, followed by dividing the square root of the output embedding size. The tanh activation function was then applied element-wise to the compatibility scores, where it was then scaled by the tanh clipping size $cl = 10$. The decoder's mask is then applied to these compatibility scores to prevent the model from assigning probabilities to invalid nodes. The compatibility score for node j at step t can be formulated as follows:

$$u_j = \begin{cases} cl.\tanh\left(r^T k_j\right) & \text{if } dm_j = False \\ -\infty & \text{otherwise} \end{cases} \tag{4.31}$$

where if the index of node j in the decoder's mask dm_j is 'False', the compatibility score is calculated otherwise if dm_j is 'True', a negative infinity value is assigned. After the compatibility scores are calculated, a softmax is applied to obtain the final log probabilities for each node. Given the current input instance X and the partial solution until step $t-1$ (represented as $\pi_{1:t-1}$), the probability of selecting a specific node x_j to visit at step t can be determined using the following formula:

$$p_\theta\left(\pi_t = x_j \,|X, \pi_{1:t-1}\right) = \frac{e^{u_j}}{\sum_{j'=0}^{tn} e^{u_{j'}}} \tag{4.32}$$

Depending on the decoding type (further explained during Section 4.4.6), the log probabilities are used to select the node for the current step. The decoding type can either be 'greedy' or 'sampling'. If the decoding type is 'greedy, ' the node with the highest probability is selected, while if the decoding type is 'sampling', a node is

randomly chosen from the probability distribution. This selected node is then added to the current partial solution.

Dynamic Encoder-Decoder

The implemented encoder and decoder dynamically updates the feature embeddings of the nodes at different steps depending on the current instance. When a node is selected from the predicted distributions generated by the decoder, that node is added to a partial solution. The partial solution is equivalent to a vehicle's route, which consists of the nodes visited by that vehicle, starting at the origin location and ending when it returns to its origin location. The remaining nodes not included in this partial solution are treated as a new instance. Solving this new instance is equivalent to constructing a new route from a different origin location. When a vehicle leaves from a different origin location, the encoder's attention mask is updated to reflect the visited nodes, focusing its attention on the unvisited nodes. The embeddings and context vectors are also updated based on the new attention mask that reflects this new state. Figure 4.6 illustrates how, after generating a partial solution (completing a driver route), a new instance is created (without the nodes of the partial solution) and used to update the feature embeddings through the encoder-decoder dynamically.

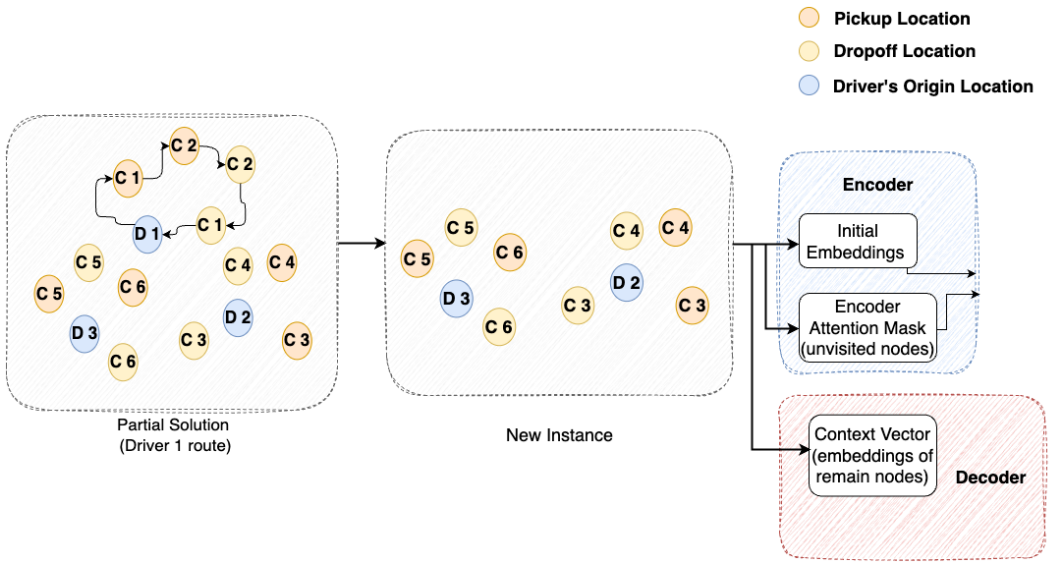


Figure 4.6 Diagram illustrating the data flow when the encoder-decoder dynamically updates the feature embeddings

The encoder's embedding h_i^t of node i at step t can be updated using the following formula:

$$h_i^t = \begin{cases} \text{ENCODER}_i^S\,(h_0^0, ..., h_s^0) & \text{if } \pi_{t-1} = u \in U \\ h_i^{t-1} & \text{otherwise} \end{cases} \tag{4.33}$$

where if node i is an origin point u, the embeddings are updated, otherwise the encoder's embeddings remains the same. The projection matrices used in the decoder's attention, such as the step context and decoder's K values, are also updated to reflect the changes in the embeddings and context vectors.

4.4.2 Actions

An action space is a set of possible actions an agent can take in a given environment. In this model, the action space consists of the possible locations that an agent can choose to visit next. There are three types of nodes in the action space: the origin, pickup, and drop-off nodes. The selected node represents the next action in the sequence of a vehicle, determining the vehicle's route. As mentioned during the decoder stage, a mask was applied to the action space to define which actions are valid for the agent to select from. The mask changes at each step, given the current state and problem constraints. The action space is represented as a discrete set of node indices. The first nu indices of the action space represent the origin locations u, the subsequent nc indices represent the pickup locations p, and the remaining nc indices represent the dropoff locations q. Given the p_i index, it's corresponding q_i index may be determined by using the following formula: $q_i = p_i + nc$.

Figure 4.7 illustrates the action space with the origin, pickup and dropoff locations.

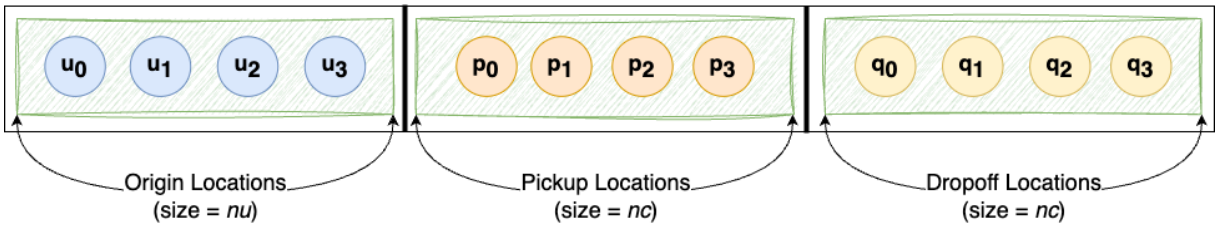


Figure 4.7 Diagram illustrating the action space

4.4.3 Reward

The reward function consisted of calculating the cost function $Cost_{total}$ (defined in Equation 4.11) and adding a penalty if any customers were left unpicked at the end of an instance. In the initial tests, the agent discovered that by picking up fewer customers, overall costs could be reduced since the agent wouldn't incur distance, waiting time, and travel time costs. This approach, however, resulted in incomplete solutions as many customers were left unpicked. To address this issue, a significant penalty was added to the reward function to ensure that the penalty for leaving customers unpicked outweighed any potential savings from not picking up customers.

The penalty for the number of customers left unpicked consists of multiplying the number of unpicked customers φ by 10000, a value that aims to magnify the penalty for leaving customers unpicked (unassigned to a driver route). The reward function can therefore be defined as:

$$R = Cost_{total} + 10000\varphi \tag{4.34}$$

4.4.4 Episodes

For each episode during training, a random dataset containing 500 instances is generated and split into batches to allow multiple instances to run concurrently. Training the model on batches increases memory efficiency and faster convergence, as the model's parameters are updated after every batch of samples, allowing it to adjust the parameters in the direction of the gradient more frequently (explained further in Section 4.4.6).

Each instance in a batch consists of randomly positioned pickup, dropoff and origin locations. In the initial state of each instance, the drivers do not have a route assigned to them, and the customers are yet to be allocated a driver. Unlike the TS experiment, in which the customers already had a driver assigned to them in the initial solution, there are no routes in the RL's initial state.

In the first step, the agent must select one of the origin locations to start a route. At each step, the agent selects a location, which can be a pickup, dropoff, or origin location, depending on the masking (explained in Section 4.4.1), which guides the agent in selecting valid actions for the current state. An agent completes a route when he returns to the chosen origin location, from which the agent must then select one of the remaining origin locations to start its next route.

An instance ends if one of the following conditions occurs: there are no more remaining customers in the instance, or there are no more drivers (origin locations) to start a new route to pick up the remaining customers. The instance's final solution represents each driver's route, which consists of the sequences of actions (locations) the agent took at each step.

After completing all instances in the current batch, the reward function (defined in Equation 4.34) then calculates the cost for each instance's final solution. For each instance, the reward function calculates the cost of each driver's route and the cost of the remaining customers left unpicked in an instance. The final cost is then used to calculate the REINFORCE loss, which involves finding the difference between the cost of the current batch and the estimated cost of the baseline model (explained further in Section 4.4.6). Depending on the gradient of the REINFORCE loss, the model's parameters are then updated. This process is repeated for all batches in the dataset. An

episode ends once all batches in the dataset are completed.

4.4.5 Hyperparameter Optimisation

Before training the REINFORCE model, Optuna, a hyperparameter optimisation framework, was used to find the optimal set of hyperparameters that minimises the reward function specified in Equation 4.11. The hyperparameters to be optimised were defined in the search space and consisted of the embedding dimension size, learning rate, and batch size.

The embedding dimension size used for the encoder and decoder was defined in the search space using the following values: 32, 64, 128, 256. The learning rate, representing the magnitude by which the parameter values are adjusted during training, was defined in the search space using the values: 0.0001, 0.001, 0.01, 0.1. When the dataset consists of many nodes (origin + pickup + drop-off locations), memory exhaustion occurs during the execution of the multi-head attention layer. Due to the GPU's memory limitations and constraints in this layer, different batch size values had to be assigned to the search space based on the number of customers in the given dataset. These values were assigned to match the tensor sizes and memory limits. Due to each dataset containing 500 samples, the batch size values defined in the search space also had to be factors of 500. The following batch size values were defined in the search space for datasets with 20 customers: 50, 100, 125. For datasets with 50 and 100 customers, the following values were defined in the search space: 20, 25, 50.

Optuna uses an objective function to determine which set of hyperparameters, defined in the search space, minimises the reward function. The objective function consists of the logic required to train the RL model using a set of suggested hyperparameters provided by Optuna during each trial. During the hyperparameter optimisation, Optuna iteratively runs trials with different sets of hyperparameters to find which hyperparameters minimise the reward function. It evaluates each set of hyperparameters using the cost returned by the objective function, which consists of the average reward of all training epochs on the given set of hyperparameters. Optuna uses the cost to evaluate the model's performance on this set of hyperparameters. Due to each trial being time-intensive because of the model's complexity, the number of trials used for the hyperparameter optimisation was set to 15. During each trial, the model was trained for 15 epochs.

The hyperparameter optimisation process was carried out for each scenario, consisting of a different number of customers and drivers. Table 4.2 shows the optimal hyperparameters achieved using Optuna for 20 customers across 4, 5, and 6 drivers, while Table 4.3 and Table 4.4 show the optimal hyperparameters achieved for 50 and 100 customers, respectively.

Table 4.2 Optimal Hyperparameters for the RL models with instances of 20 Customers

20 Customers	Drivers		
	4	5	6
Embedding Dimension Size	256	64	64
Learning Rate	0.0001	0.0001	0.0001
Batch Size	50	100	100

Table 4.3 Optimal Hyperparameters for the RL models with instances of 50 Customers

50 Customers	Drivers		
	4	5	6
Embedding Dimension Size	32	128	128
Learning Rate	0.0001	0.0001	0.0001
Batch Size	20	50	50

Table 4.4 Optimal Hyperparameters for the RL models with instances of 100 Customers

100 Customers	Drivers		
	4	5	6
Embedding Dimension Size	128	32	64
Learning Rate	0.0001	0.0001	0.0001
Batch Size	25	20	20

4.4.6 Training the REINFORCE Algorithm

The REINFORCE algorithm is a policy gradient algorithm that aims to lower costs (higher reward) by adjusting the policy's parameters. For this model, the REINFORCE algorithm was trained with a baseline model. The purpose of this baseline model was to estimate the expected cost associated with taking a sequence of actions under the model's current policy. The baseline serves as a reference point for comparison against the actual cost. The difference between the actual cost achieved from the trained model and the baseline value determines the REINFORCE loss. The gradient of this loss is used to adjust the model's parameters, increasing the likelihood of actions that lead to higher rewards. These likelihoods represent the probabilities that the agent will

take each possible action in a particular state. This baseline model can help reduce the variance in the policy gradient estimates.

The baseline model was initialised by creating a model based on the initial training model's parameters. A new dataset was then generated for the baseline model. The baseline model was evaluated on this dataset using a greedy decoder type to calculate the average cost of the current baseline model. During each epoch, a new random dataset was created using 500 samples, each including nu number of origin nodes, nc number of pickups and nc number of drop-off nodes.

During the initial training epochs, the baseline model goes through a warm-up phase, where the model is gradually introduced to the dataset to avoid potential issues like instability or divergence that might arise from immediate exposure to the entire dataset. The alpha parameter controls the warm-up phase. If the parameter is below 1, the model is in the warm-up phase, and therefore, the model skips the computation of baseline values. In contrast, if the alpha parameter is above 1, the warm-up period is completed, and the baseline values are calculated for each sample in this dataset using the decoder set to 'greedy' for a greedy execution of the baseline model. These baseline values are used further on to compare performance and reduce the variance of the REINFORCE's policy gradient estimator.

Each epoch's dataset is split into batches using an optimal batch size specified during the parameter optimisation, as explained previously in Section 4.4.5. The REINFORCE algorithm evaluates each batch separately, using each batch's instances as input data. The model evaluates each batch by iterating through the decoding steps until the state of each instance in the batch is marked as finished (based on the conditions specified in Section 4.4.4). For each instance in the batch, the model then calculates the total costs of the final solution and the log-likelihood of the selected actions.

To calculate the REINFORCE loss for the current batch, we first determine the baseline values using the current batch as input and the total costs achieved from evaluating the model on the current batch. If no baseline values are precomputed, the baseline values are calculated based on three conditions. If the alpha parameter is 0 (warm-up has not started), the baseline values are the exponential moving average (EMA) cost of the previous batches; if there are none, it is the mean of the current batch's cost. If the alpha parameter is less than 1 (warm-up phase), the combination of the EMA and baseline cost is calculated. If the alpha parameter equals 1 (warm-up phase completed), the baseline value is the combination of the EMA set to 0 and the baseline cost. The baseline values are excluded from the gradient computation to prevent any gradient from flowing through them during backpropagation.

The REINFORCE loss is then calculated by finding the difference between the total cost (achieved from evaluating the model) and the estimated baseline value. The

difference is then multiplied by the log-likelihood obtained earlier to weigh the probability of each action being chosen by the model's policy. The REINFORCE loss represents how well the current policy performed on a given set of inputs relative to the baseline estimate. The gradient of the REINFORCE loss was calculated with respect to the model's trainable parameters, which consisted of the weights of the dense layers used earlier for the encoder and decoder. The gradient of the REINFORCE loss determines the direction and magnitude of adjustments needed for each parameter in the model to improve its performance relative to the baseline. Once the gradients of the REINFORCE loss are calculated, they are then passed to the Adam optimizer, which is an extension of the gradient descent method. The Adam optimizer applies the gradients to their corresponding model's parameters to improve the policy performance.

After training the model on all batches, the model is evaluated on the baseline dataset to determine the mean cost of all instances. The difference between the mean cost of the training model and the mean cost of the baseline model is then calculated to determine if the training model performed better or worse than the baseline model. If the difference is negative, this indicates that the training model performed better. A statistical test to confirm the significance of improvement in the trained model is carried out, and if the improvement is significant, the baseline model is updated. The baseline model is updated by copying the weights from the trained model to create a new baseline model. A new dataset for the new baseline model is also created to prevent possible overfitting. The new baseline model is evaluated on this dataset using a greedy decoder to determine a new mean baseline cost.

The above training procedure is repeated for 100 epochs, allowing the REINFORCE algorithm combined with the baseline model to optimise its parameters and find a policy that minimises overall costs. The policy represents the model's ability to generate optimal solutions for the MVRRPP based on the given data. For each scenario containing a different number of customers and drivers, a separate model was used for training using the respective dataset of that scenario, as defined in Section 4.2.

4.4.7 Comparing REINFORCE with Tabu Search

The trained REINFORCE models were evaluated on their respective validation sets. These validation sets are the same datasets used to evaluate the TS algorithm on scenarios with different number of customers and drivers. Each instance (sample) in the validation set represents a different MVRRPP instance on which the trained model is tested on. A final solution was generated when testing the trained RL model on each instance in the validation set. The final solution consisted of the sequences of actions taken in the environment at each step for that corresponding instance. It represents all

the driver routes and the order in which the customers' pickup and drop-off locations were visited. The performance of the trained model was evaluated using the final solutions generated for each instance in the validation set. The metrics for evaluating each instance's final solution consisted of the waiting time, travel impact time, driver's distances, and time taken to solve the instance. For each scenario of customers and drivers specified in Section 4.2, the metrics achieved using RL were compared to the results achieved using TS on the same dataset.

When evaluating RL and TS experiments in different scenarios, the RL experiment used the pre-trained model to find solutions for unseen scenarios without needing to retrain. In contrast, TS must start the search process for each unseen scenario from the beginning.

4.5 Objective 3: Evaluate the effect of combining a RL approach with optimisation based on Tabu Search

The third objective of this research consisted of using the output achieved in the second objective as the initial solution for the TS. This objective aims to determine whether combining RL with TS can enhance the optimisation process and achieve better results in terms of solution generation time and quality.

While the initial solution in Experiment 1 was generated by distributing the customers equally across each driver's route, in this experiment, the RL solution is used as the initial solution for the TS to generate the initial neighbourhood solutions. Figure 4.8 illustrates the data flow of the TS with RL model.

As previously mentioned in Section 4.4.7, the RL solution obtained from the previous experiment includes the sequence of visited locations for each driver's route, achieved after testing the trained RL model on the validation set.

For this experiment, the TS algorithm used the same tabu list size and was executed on the same number of iterations as Experiment 1. To evaluate the impact of using the RL solution as the initial solution, TS was evaluated on the same dataset used to evaluate the algorithms in the previous experiments. By using the same dataset, iterations, and tabu list size, we were able to compare the metrics for all the experiments under the same conditions.

This process was repeated for each scenario of customers and drivers (specified in Section 4.2), where their respective RL solution was used as the initial solution and evaluated using TS separately on the dataset of that scenario.

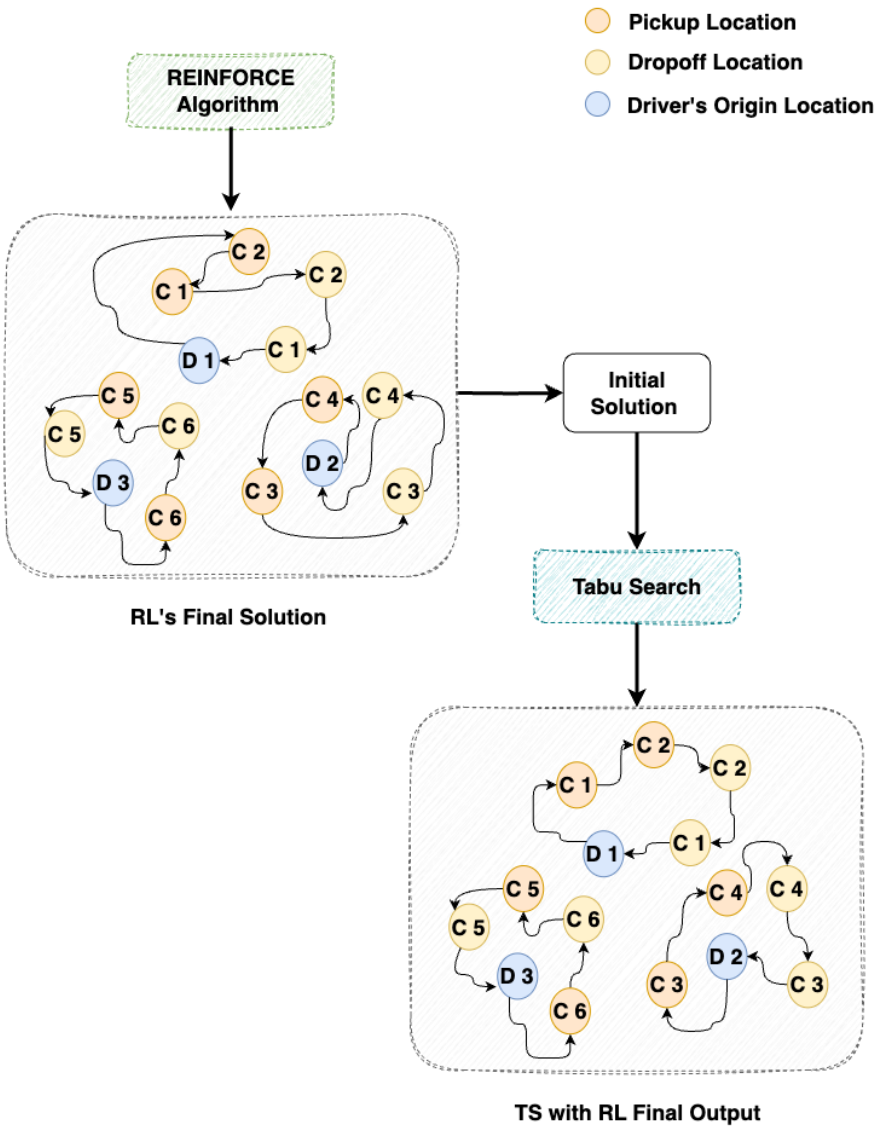


Figure 4.8 Diagram illustrating the data flow of the Tabu Search with Reinforcement Learning experiment

4.6 Software Libraries and Hardware Used

The TS and RL models were implemented using Python 3.9.16. The RL model was built using Tensorflow, a machine-learning library used for training deep neural networks. Table 4.5 shows the packages used for TS and RL implementations and their version.

The parameter optimisation and training of the RL model were run on a Paperspace Linux virtual machine using an 8 virtual-core Intel Xeon Gold 5315Y CPU, 45 GB of RAM, and an NVIDIA RTX A4000 with 16GB of graphics memory.

Table 4.5 Packages used for TS and RL

Packages	**Version**
numpy	1.23.4
tensorflow	2.9.2
keras	2.9.0
optuna	3.4.0
pandas	1.5.0
matplotlib	3.6.1
scipy	1.9.2
plotly	5.18.0
seaborn	0.12.0
tqdm	4.64.1

Once the training of the RL model was complete, the trained model was evaluated on the validation set using a MacBook Air with an 8-core Apple M1 CPU, 8 GB of RAM, and an integrated 7-core GPU. Both TS implementations, one using an equally distributed initial solution and the other using RL solution as the initial solution, were run on this same hardware.

5 Results & Evaluation

This chapter provides the results and evaluation of the three experiments described in the previous chapter. It includes an overview of the objectives defined for this research, an overview of the metrics used for evaluation, and sections for each objective that describe and evaluate each experiment's results.

The first objective of this research involved implementing a TS algorithm as a baseline to solve the MVRRPP and evaluating its performance. The second objective involved modelling the MVRRPP as an RL problem by defining the state, action, and rewards and comparing its performance with the TS algorithm. The third objective involved using the output achieved from the RL algorithm (during the second objective) as the initial solution for the TS algorithm and evaluating its impact on performance when finding the optimal solution for the MVRRPP. The performance of each objective was evaluated using the following five metrics:

1. the waiting time, which is the time it takes for a vehicle to visit a customer's pickup location
2. the travel impact time, which is the difference between the time it takes to drop off a customer from their pickup location and the optimal time from the customer's pickup to the customer's dropoff location without visiting any other customer in between (without ride-pooling)
3. the drivers' distances, which is the total distance travelled in a route by each driver.
4. the running time, which is the total time to solve an MVRRPP instance (a sample in the dataset) when using RL or TS
5. the total number of iterations needed to find the optimal solution when using TS

The travel impact time metric was used to indicate the additional time incurred by the customer due to ride-pooling, from when they are picked up to when they are dropped off. For a quantitative measure, the waiting time and travel impact time metrics represent distance as a base unit, where every 1 unit represents 1 km. The time a customer waits for a vehicle to pick them up is equivalent to the distance the vehicle needs to travel until it reaches the customer's pickup location. The longer the distance a vehicle travels, the longer a customer has to wait for pickup.

Each experiment was evaluated with 4, 5, and 6 drivers, each containing 20, 50, and 100 customers, for a total of 9 datasets. For each group of customers (20, 50, 100), different locations were used for the datasets containing 4, 5 and 6 drivers. Each

dataset consisted of 500 different MVRRPP instances. The following sections will provide the results obtained for each experiment and an evaluation of these results.

5.1 Objective 1 - Tabu Search

For this experiment, the TS algorithm was evaluated on datasets involving different numbers of customers and drivers, with each dataset containing 500 problem instances. TS was evaluated on each instance using an initial solution consisting of an equal distribution of customers across all drivers. This initial solution was used as the starting point for the search process.

Figure 5.1 illustrates an instance from the dataset containing 20 customers and 4 drivers while Figure 5.2 illustrates the final solution of this instance after using TS. The graphs show the locations of each driver's origin location, represented by the green points, and each customer's pickup and dropoff locations, represented as the slate blue and red points, respectively. The final solution of the instance, as depicted in Figure 5.2, shows each driver's route, represented as a line with arrows showing the sequence in which the nodes (locations) were visited by that vehicle. Figure 5.3 highlights each of the four drivers' routes, with each route represented as a different colour. The graphs show how each route starts from its origin location, followed by the customers' pickup and dropoff locations (with the customer's pickup being visited before its respective dropoff), and lastly, ending the route by returning to the origin location.

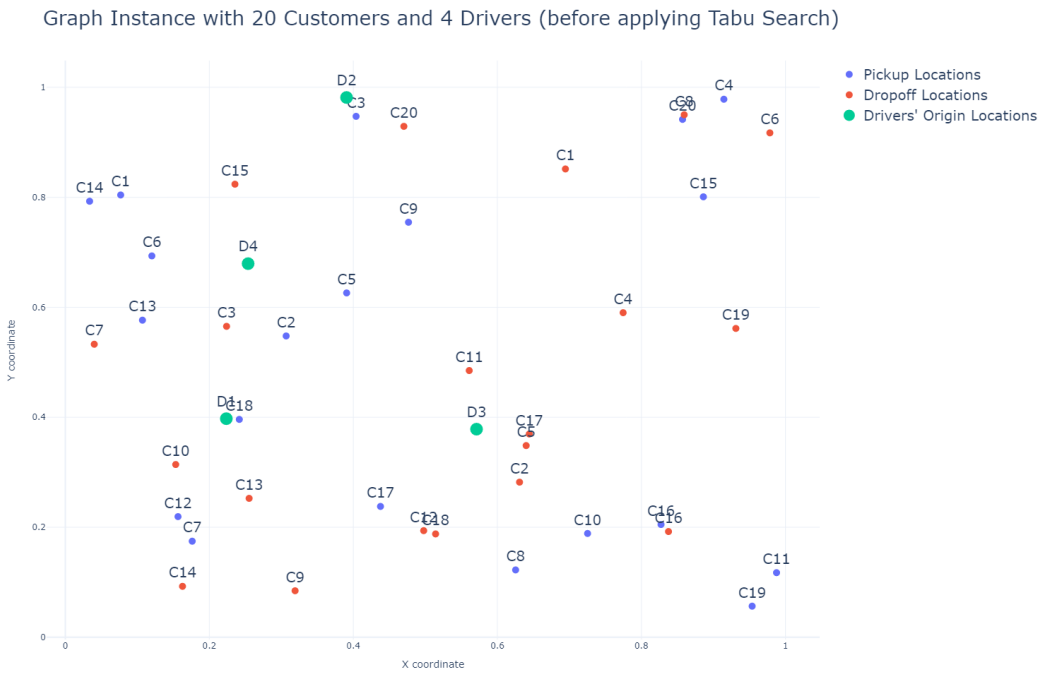


Figure 5.1 Graph Instance with 20 Customers and 4 Drivers before applying TS

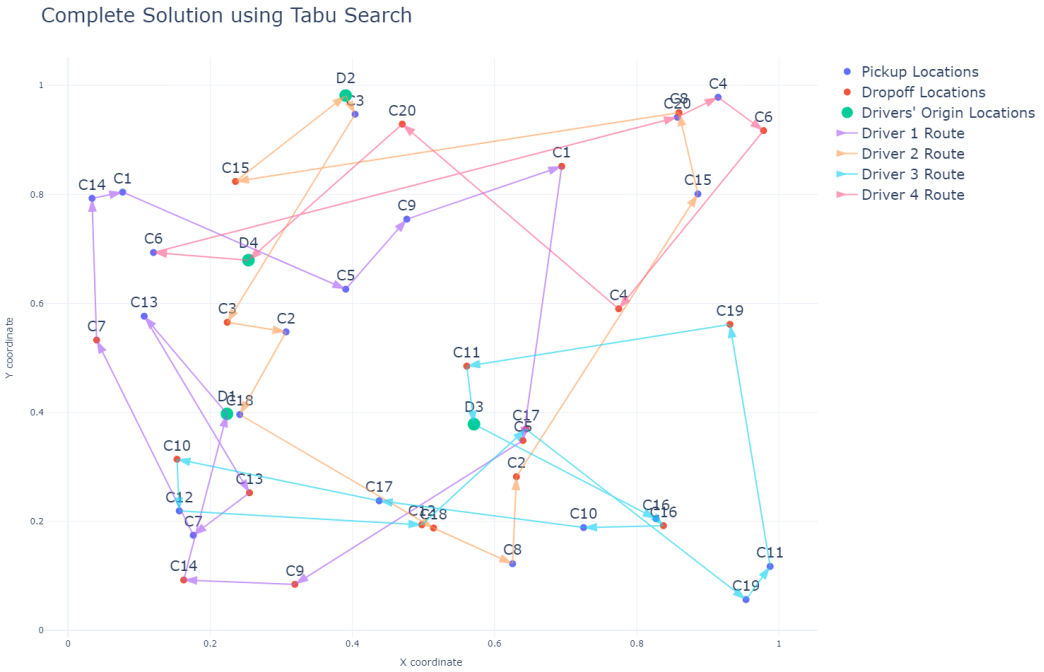


Figure 5.2 Final Solution after applying TS on the graph instance of 20 customers and 4 drivers

(a) Driver 1 Route

(b) Driver 2 Route

(c) Driver 3 Route

(d) Driver 4 Route

Figure 5.3 Drivers' Routes obtained using TS on an instance with 4 drivers and 20 customers

Figures 5.4 to 5.8 show the spread and distribution of the data. Each box plot shows a box representing the interquartile range that indicates where the middle 50% of the data lies. The box's lower and upper edges represent the first and third quartiles (with each quartile representing 25% of the data). The horizontal line inside the box represents the median of the data. The interquartile range indicates how the samples in the data are dispersed. The lines extending vertically on each box from the quartiles represent the 'whiskers' and indicate the data's minimum (lower line) and maximum (upper line) values. These whiskers provide insight into the range of the data. The data points outside the whiskers indicate the outliers within the data.

Figure 5.4 shows a box plot with the waiting times across 500 samples for each driver instance, consisting of 20, 50, and 100 customers. As the number of drivers increases with the same number of customers, both the minimum and maximum range of waiting times tend to decrease, suggesting a reduction in the spread of waiting times as more customers are distributed amongst drivers, reducing the overall waiting time as more customers can be attended to by different drivers. The interquartile range also becomes slightly narrower, suggesting that the variability in waiting times diminishes as it is more concentrated around the median. However, when comparing the waiting time across the same number of drivers but with additional customers, the interquartile range increases, indicating a higher variability along the centre of the box plot among each group of customers. The minimum and maximum range of waiting times also increases with the increase in customers, indicating a broader spread of waiting times across different customers. This broader spread indicates the increasing complexity and challenges when finding optimal routes with more customers, as additional locations must be visited, increasing the overall waiting time.

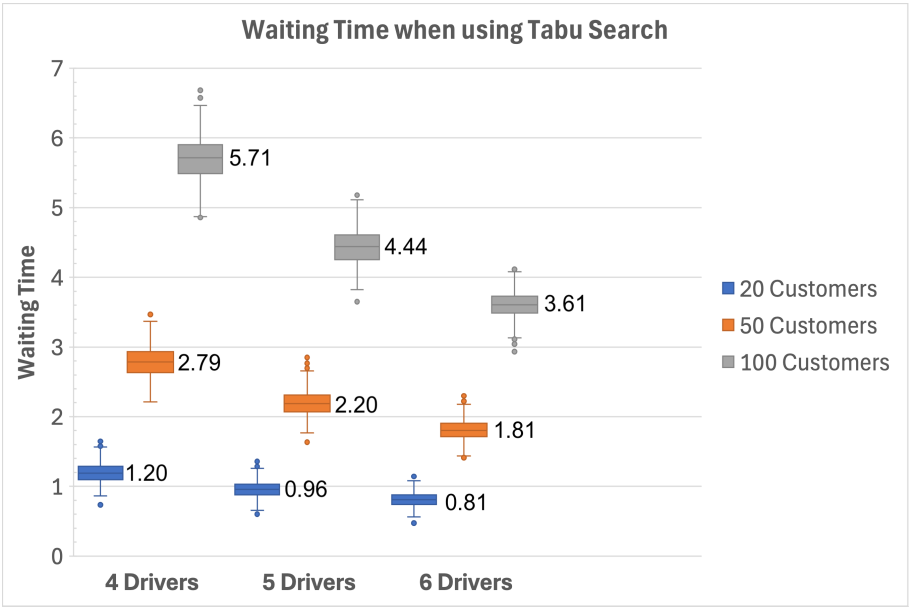


Figure 5.4 Box Plot with Waiting Time across different instances when using TS

Figure 5.5 illustrates a box plot showing the travel impact time on different driver instances with various customer groups. For each box plot belonging to the same number of customers, the interquartile range narrows slightly as more drivers are added, suggesting less variation in the travel impact time for the middle (50%) of the data as the drivers increase. Each box plot's minimum and maximum range also reduces as the drivers increase within the same customer group, indicating a smaller spread between the shortest and longest travel impact time in the data as the drivers increase. This shows that as the number of drivers increases, the travel impact time is reduced as the customers' demand is being shared across more drivers' routes. However, when comparing the box plot of 6 drivers with that of 5 drivers in instances of 100 customers, there was a slight increase in the box plot's interquartile range and the minimum and maximum range. This suggests a small increase in variability in the travel impact time on the box plot with 6 drivers. The box plots also indicate an increase in the median travel time as more customers are added to an instance of the same number of drivers. This is likely due to the driver making more stops along the route to pick up or drop off additional customers, increasing the time a customer travels between pickup and destination.

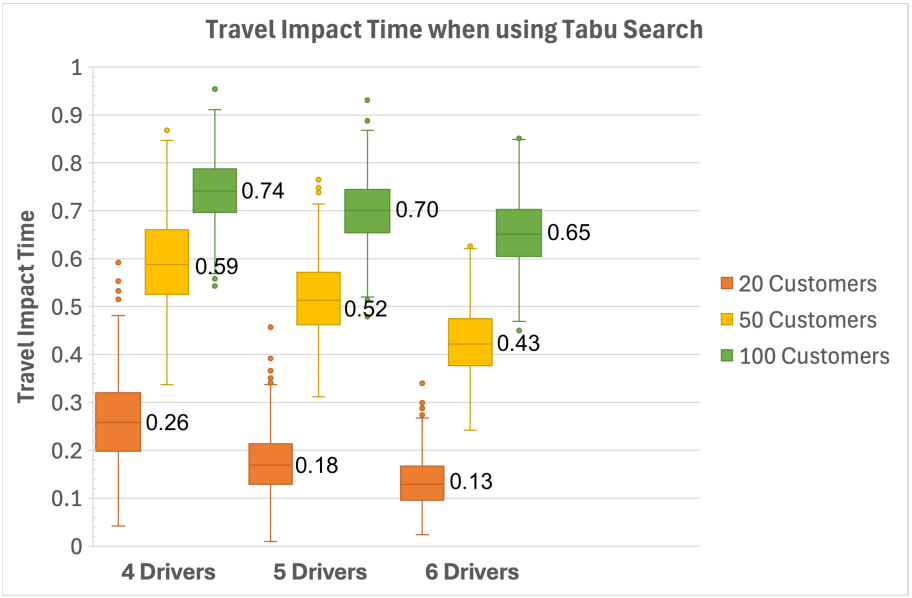


Figure 5.5 Box Plot with Travel Impact Time across different instances when using TS

A box plot illustrating the driver distances in instances with different drivers and customers is shown in Figure 5.6. The box plot shows that the interquartile range and minimum and maximum range decrease as the number of drivers increases in instances of the same number of customers. The decrease in the minimum and maximum range shows a reduction in the spread of distances, whilst the decrease in the interquartile range shows that the variability within the central portion was reduced. As the number of drivers increases, the average total distance decreases as more customers are being

picked up by different vehicles, decreasing the number of stops a vehicle must make. The box plots also indicate that as customers increased within instances of the same number of drivers, the median values of the box plots also increased, along with the interquartile range and minimum range. This shows how the distance travelled by the drivers increases as the number of customers increases, as the driver has to make additional stops along the route for further pickups and dropoffs. Each additional stop adds distance to the overall route as the vehicle must travel from one stop to the next.

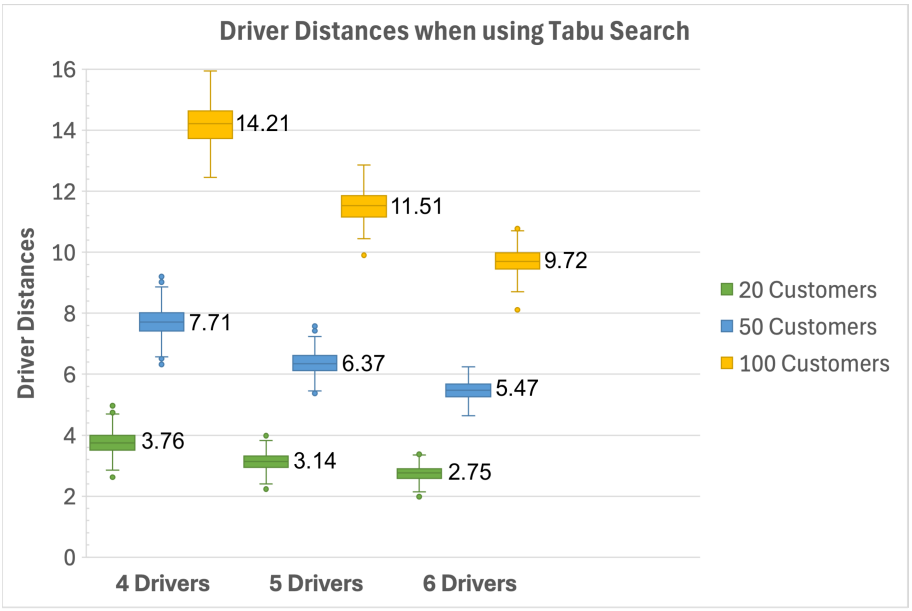


Figure 5.6 Box Plot with Drivers' Distances across different instances when using TS

Figure 5.7 illustrates a box plot with the number of iterations TS took to generate solutions across varying numbers of drivers and customers. The box plot shows that the median values and the size of the box plots remain consistent across different numbers of drivers with the same customers. As the number of customers increased within instances of the same number of drivers, the min-max range and median also increased. This indicates that the solution space was becoming more complex and challenging and required more iterations to converge.

Figure 5.8 shows box plots with the running time taken to generate solutions using TS for each group of customers across multiple drivers. Within instances of the same number of customers, the running time increased slightly as the number of drivers increased; this is most noticeable in instances with 50 and 100 customers. The slight increase in running time between drivers indicates that the complexity of the problem expanded as more driver routes needed to be considered, requiring the TS to take longer computation times to find the local optimal solution. The box plots also indicate a substantial increase in running time when adding additional customers, as indicated by the box plot's minimum and maximum range and median, suggesting that the TS struggles to handle large instances within a reasonable time frame.

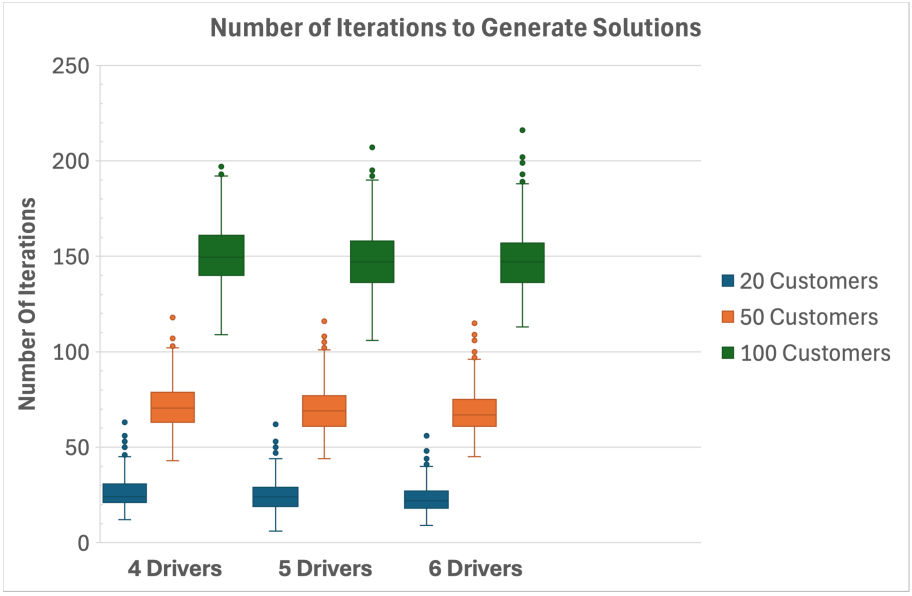
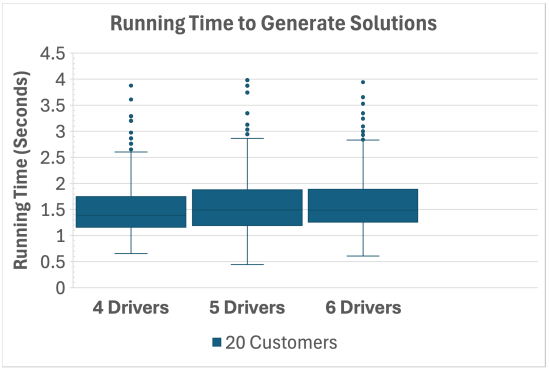
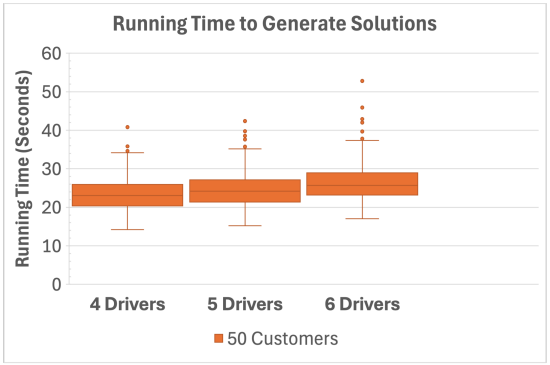


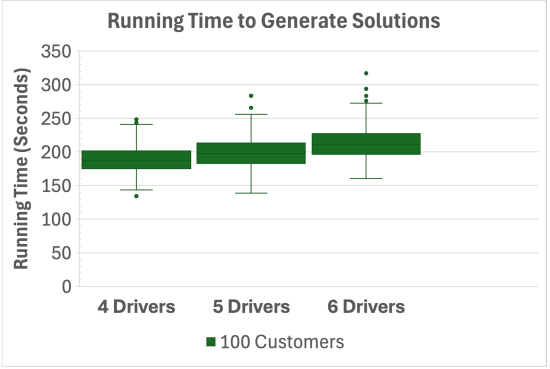
Figure 5.7 Number of Iterations to generate solutions when using TS



(a) 20 Customers



(b) 50 Customers



(c) 100 Customers

Figure 5.8 Running Time to generate solutions when using TS

Figures 5.9 to 5.11 illustrate the relationship between the number of iterations and the time taken to find a solution for each instance in the dataset. Each scatter plot represents instances of 20, 50, and 100 customers, showing the number of iterations and the time TS takes to find the 'best solution' for each instance in the datasets containing 4, 5 and 6 drivers. The scatter plots show the algorithm's complexity; as the number of drivers and customers increases, the number of possible solutions increases, and the search space and neighbourhood becomes more larger. Therefore, converging to a local optimum takes more iterations, leading to increased computational time to find a solution, as apparent in each scatter plot with the increase in drivers.

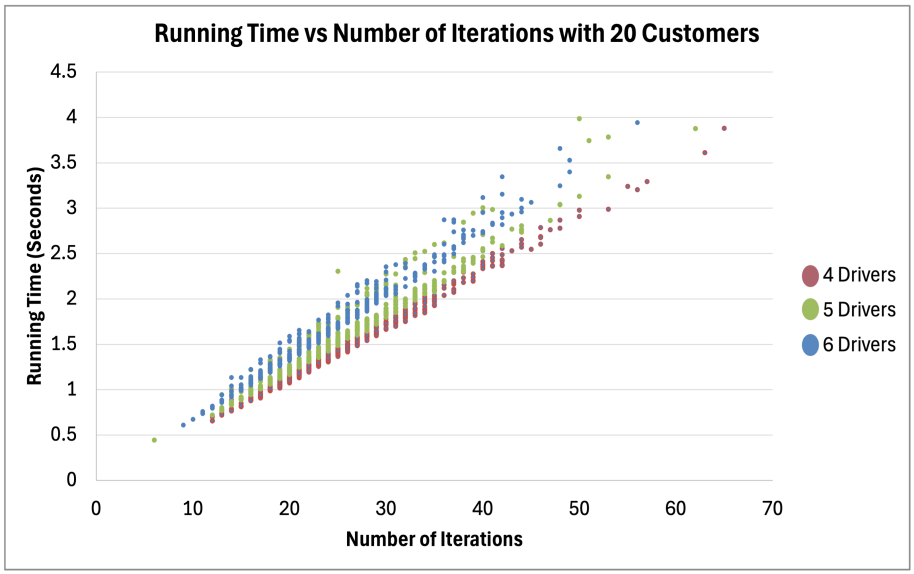


Figure 5.9 Running Time vs Number of Iterations when using TS with 20 Customers

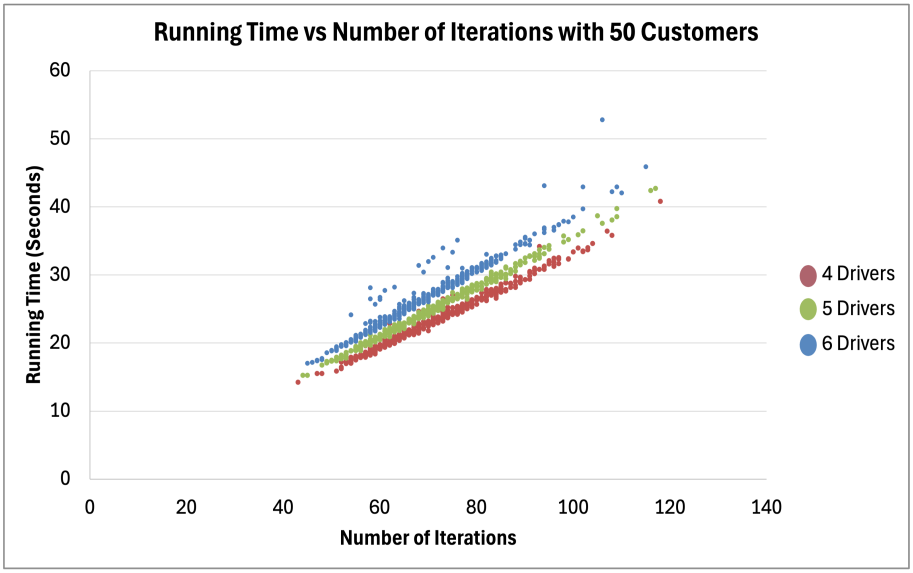


Figure 5.10 Running Time vs Number of Iterations when using TS with 50 Customers

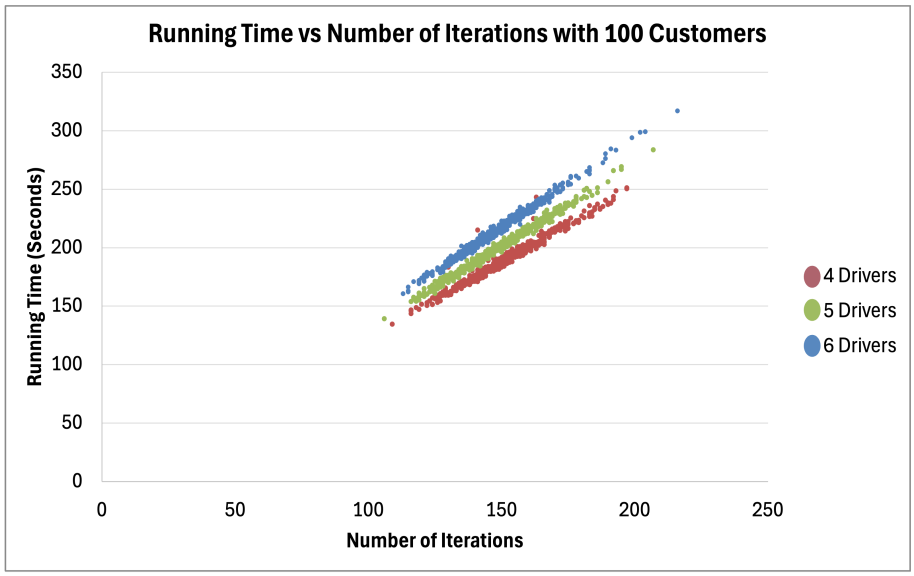


Figure 5.11 Running Time vs Number of Iterations when using TS with 100 Customers

5.2 Objective 2 - The Reinforcement Learning Model

The second experiment involved evaluating the performance of the trained RL models across instances with different numbers of customers and drivers. For comparison purposes, the RL models were evaluated using the same dataset that was used to evaluate the TS algorithm.

Figure 5.12 displays a graph depicting an instance from the dataset consisting of 20 customers and 4 drivers, while Figure 5.13 shows the final solution of this instance after using RL. Figure 5.14 illustrates each driver's route as separate graphs.

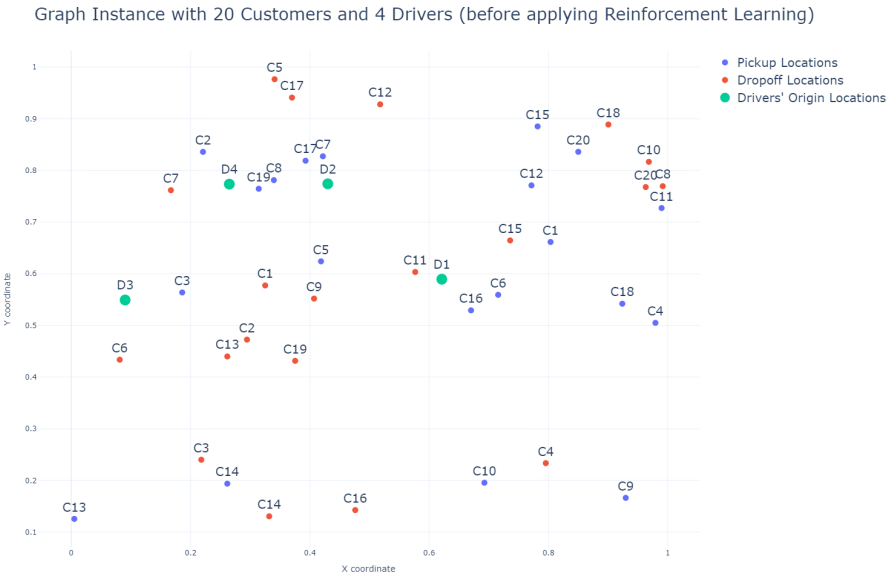


Figure 5.12 Graph Instance with 20 Customers and 4 Drivers before applying RL

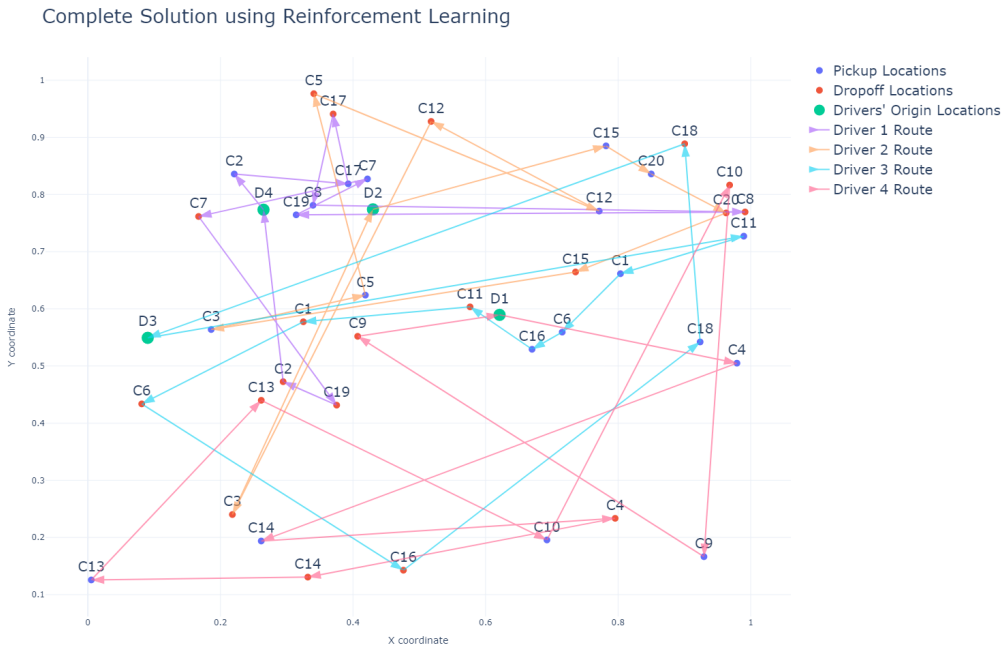
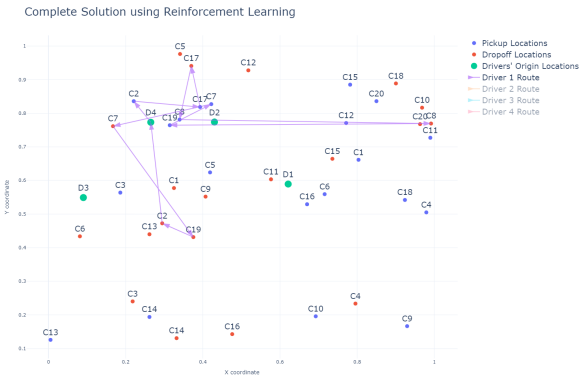
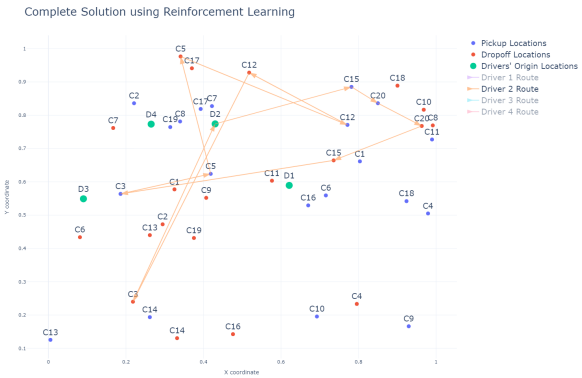


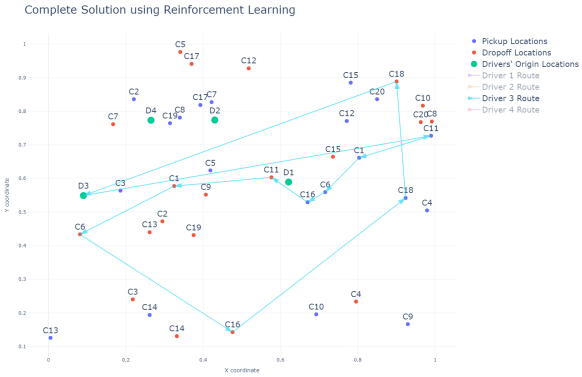
Figure 5.13 Final Solution after applying RL on the graph instance of 20 customers and 4 drivers



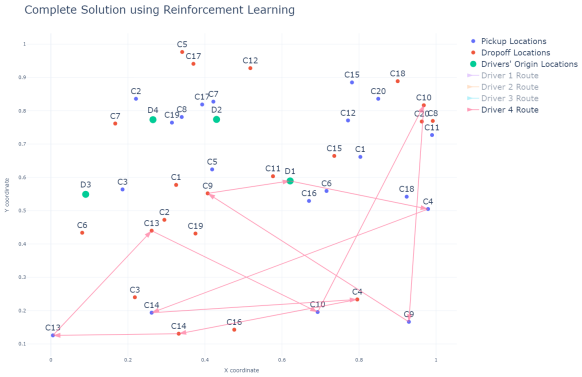
(a) Driver 1 Route



(b) Driver 2 Route



(c) Driver 3 Route



(d) Driver 4 Route

Figure 5.14 Drivers' Routes obtained using RL on an instance with 4 drivers and 20 customers

Figure 5.15 illustrates a bar chart with the average customer's waiting time when using RL across different instances of customers and drivers. The bar charts show that the average waiting time increases as the number of customers increases in instances with the same number of drivers. This is noticeable in instances with 100 customers as the waiting time significantly increases. As the number of customers increases, the driver has to visit more locations for pickup and drop off, increasing the overall waiting time of customers. When comparing the performance across different numbers of drivers with the same amount of customers, the waiting time varies and appears to be non-linear, with only a few instances indicating a linear improvement in waiting time as the number of drivers increases.

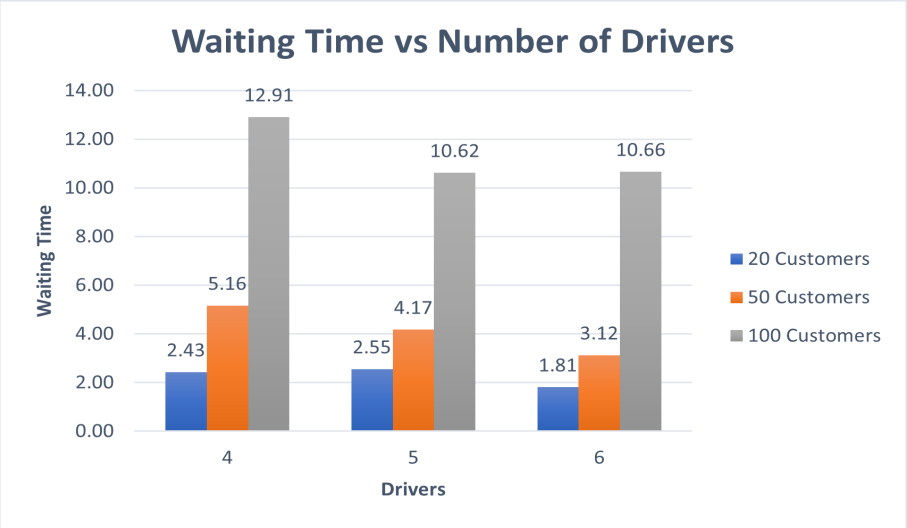


Figure 5.15 Average Waiting Time vs Number of Drivers when using RL

Figure 5.16 illustrates a box plot displaying the waiting time performance when using RL across all instances. The box plot shows that as customers increase, the box plot position moves further up, showcasing an increase in waiting time.

Figure 5.17 shows a bar chart with the average travel impact time across different RL instances. The bar chart indicates that the travel impact time increases as more customers are added to a driver instance. When comparing instances between 4 and 5 drivers, the travel impact time across each customer group decreased. However, when comparing instances between 5 and 6 drivers, the travel impact time increased in instances of 20 and 50 customers, whilst in instances with 100 customers, the average travel impact time remained the same, suggesting that the additional driver did not improve performance.

Figure 5.18 shows a box plot with the travel impact time performance across all instances using RL. The box plot shows a similar performance between 20 and 50 customers, with the travel impact time increasing exponentially with instances of 100 customers.

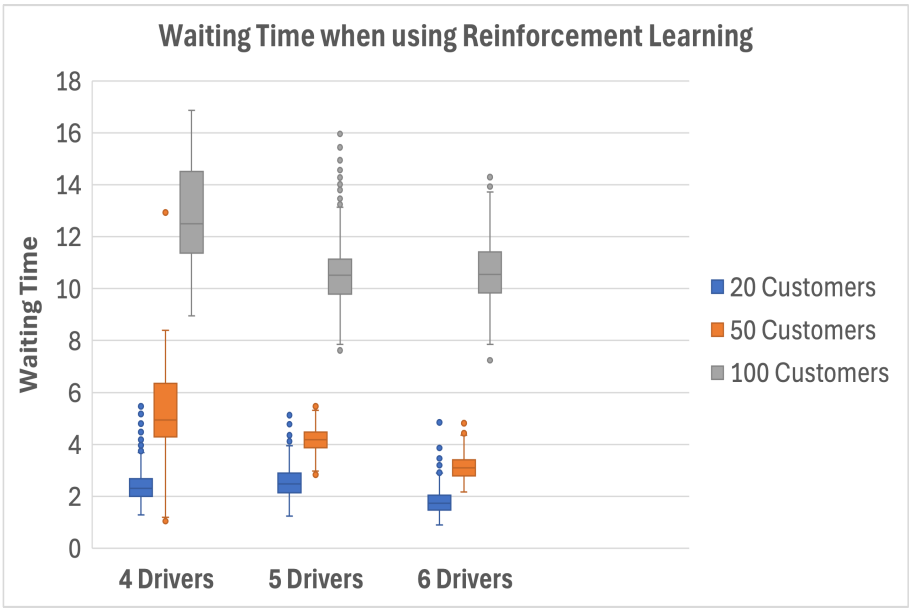


Figure 5.16 Box Plot with Waiting Time across different instances of drivers and customers using Reinforcement Learning

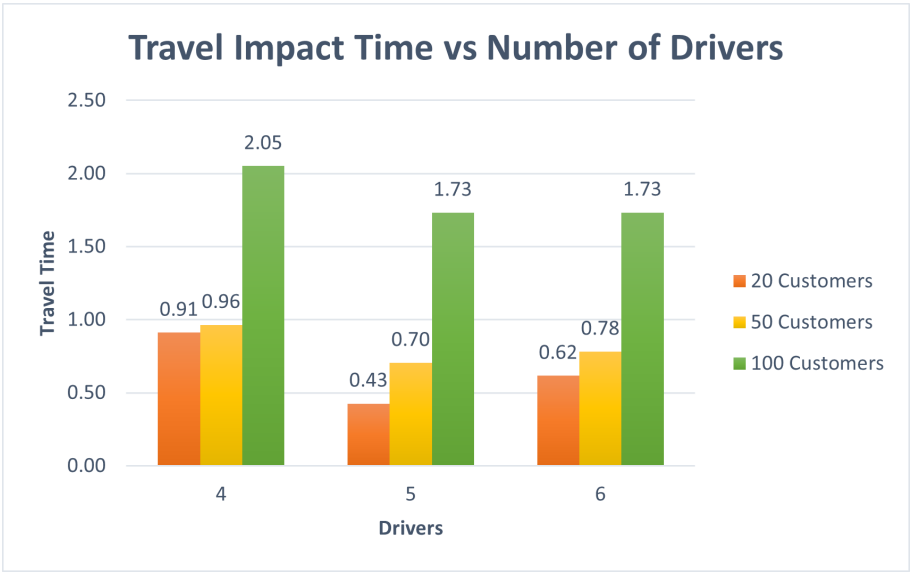


Figure 5.17 Average Travel Impact Time vs Number of Drivers when using RL

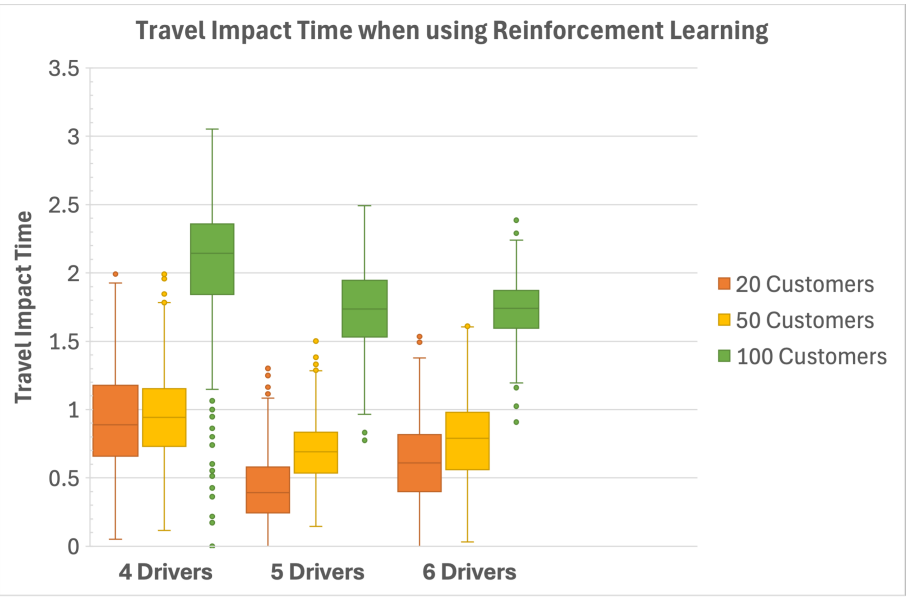


Figure 5.18 Box Plot with Travel Impact Time across different instances of drivers and customers using Reinforcement Learning

The increase and minimal impact in waiting time and travel impact time on certain instances when adding more drivers indicate RL's challenges in finding the trade-off between waiting time and travel impact time. As the number of drivers increases, the RL model attempts to balance the allocation of customers among these drivers to minimise overall waiting time and travel impact time. However, with increased drivers, RL may distribute customers among drivers less efficiently, resulting in longer times. The RL model seeks to find the balance where waiting time is minimised without increasing the customer's travel impact time.

The bar chart presented in Figure 5.19 shows the average total distance travelled by the drivers in different instances using the trained RL model. For each driver, the distance travelled increased as more customers were added. In contrast, the bar chart shows that the distances decreased as more drivers were added to instances with the same number of customers. The decrease in distance as more drivers are added to an instance suggests that the RL model is effectively optimising the order in which customers are visited by assigning pickups and drop-offs based on proximity and effectively distributing customers across routes to minimise the overall distance travelled.

Figure 5.20 exhibits a box plot showing the distance performance when using RL. The box plot confirms that the distances increase as the number of customers increases across all driver instances. It also confirms that the distances decrease across instances with the same number of customers when adding more drivers.

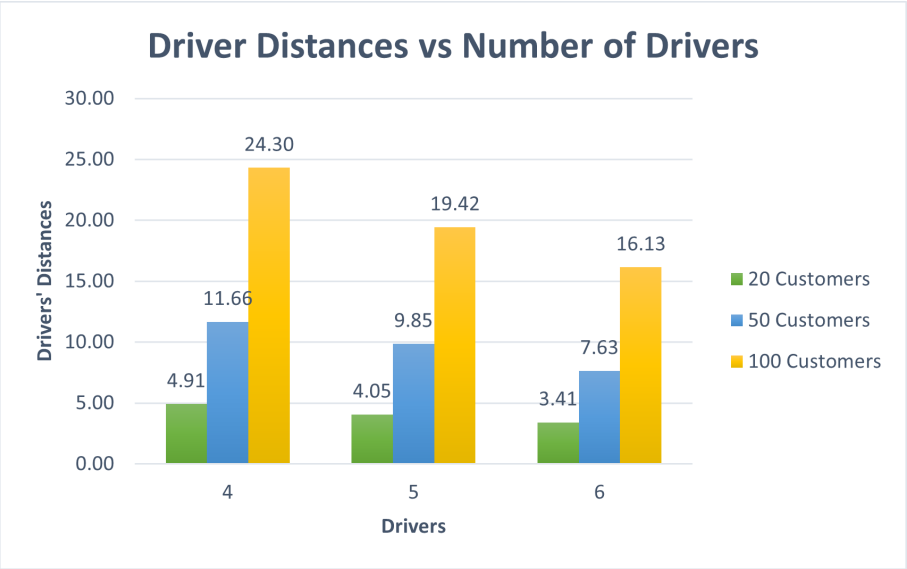


Figure 5.19 Average Driver Distances vs Number of Drivers when using RL

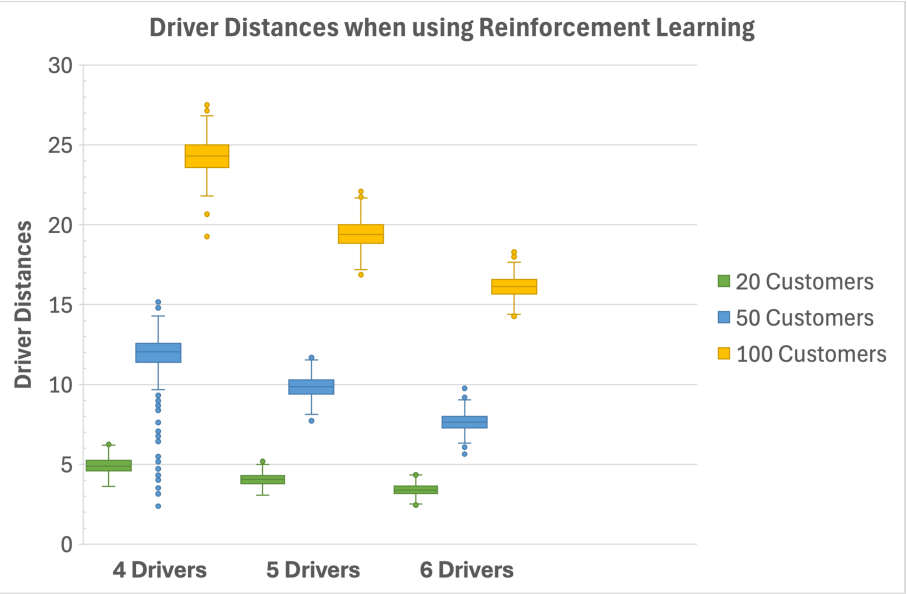


Figure 5.20 Box Plot with the drivers' distances across different instances of drivers and customers using Reinforcement Learning

The bar chart in Figure 5.21 shows the average running time for the RL to complete each solution. The running time increases as additional customers are added and remains consistent with the addition of drivers. The increase in running time shows how, as the number of customers increases, the problem complexity grows exponentially, and the RL model needs additional time to consider more possible combinations.

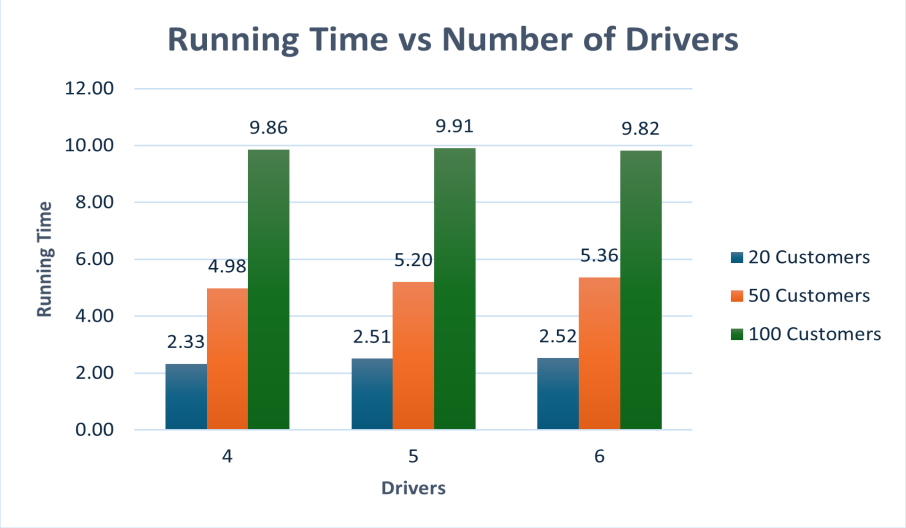


Figure 5.21 Average Running Time vs Number of Drivers when using RL

The box plot in Figure 5.22 shows RL's running time performance across each instance. It indicates that it takes longer to complete solutions as the number of customers increases; however, the running time remains consistent as the number of drivers increases across instances with the same number of customers.

Figure 5.22 Box Plot with the Running Time across different instances of drivers and customers using Reinforcement Learning

Figures 5.23 to 5.25 show the total average waiting time for both TS and RL experiments on instances with 20, 50 and 100 customers across different number of drivers. Lower bars indicate a better performance as they represent shorter waiting times. The bar charts show that the TS performed better in terms of waiting time for each customer group across each driver instance.

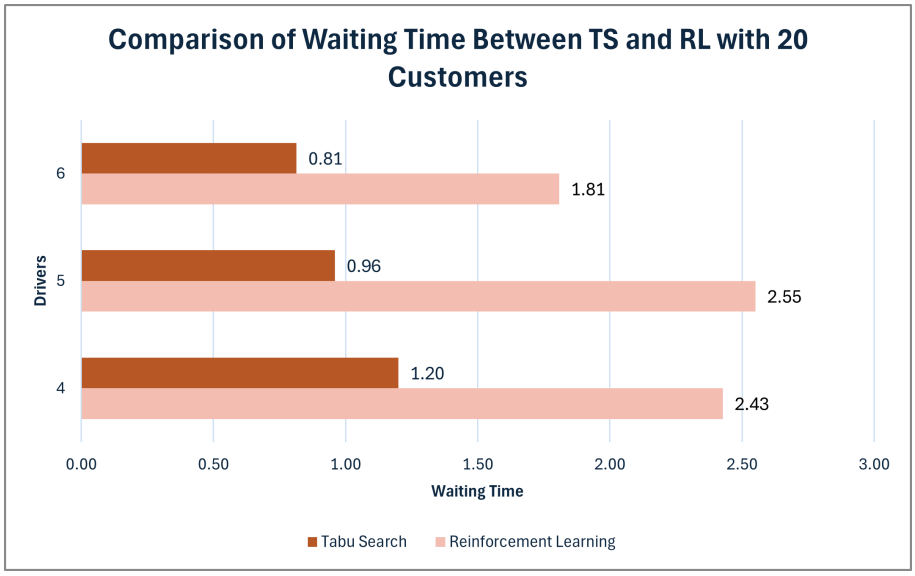


Figure 5.23 Comparing Average Waiting Time between TS and RL with 20 customers across each driver instance

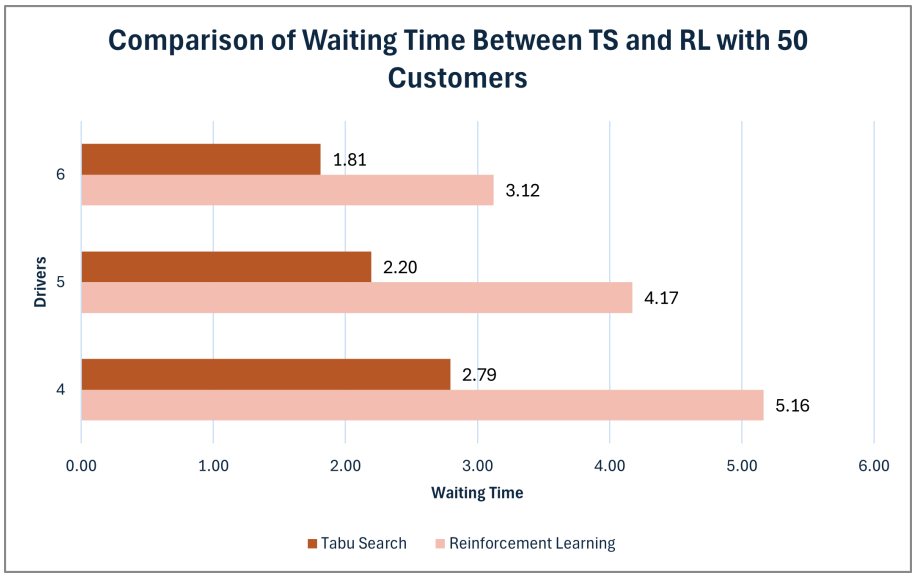


Figure 5.24 Comparing Average Waiting Time between TS and RL with 50 customers across each driver instance

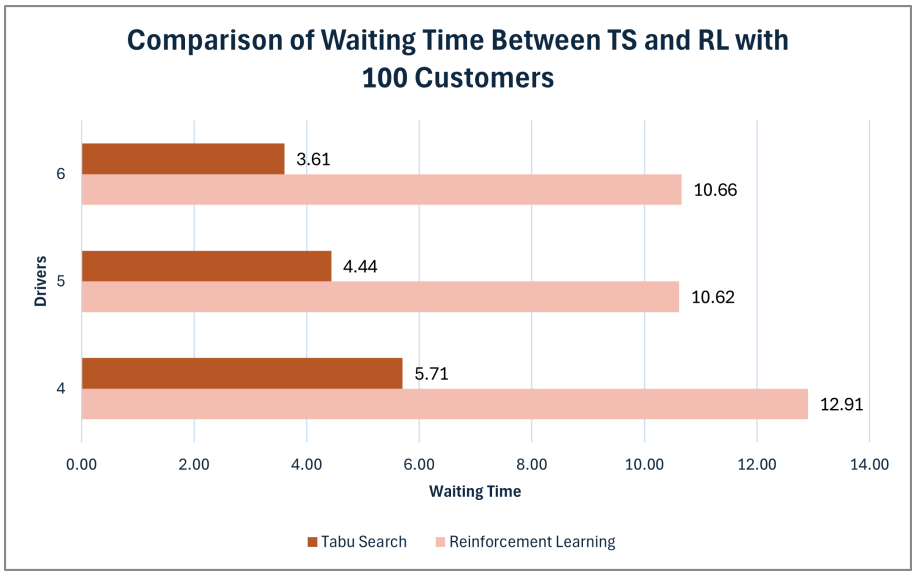


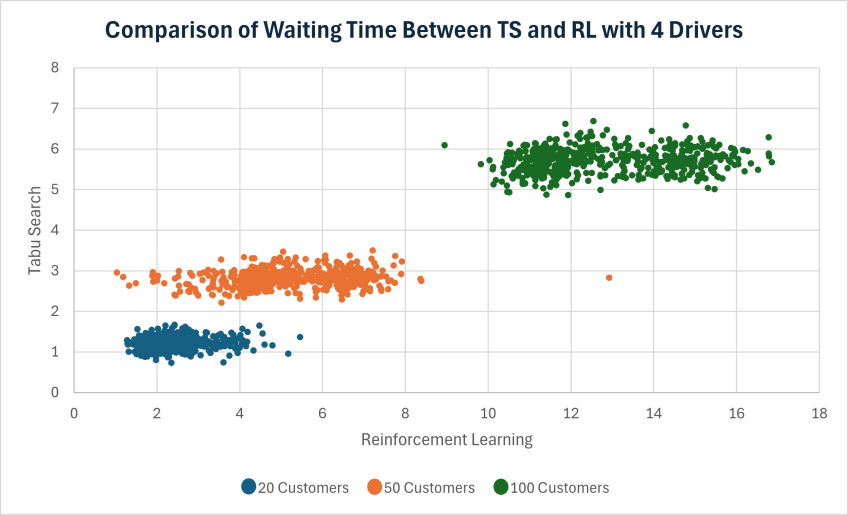
Figure 5.25 Comparing Average Waiting Time between TS and RL with 100 customers across each driver instance

Figure 5.26 shows the scatter plots comparing the waiting time performance of TS and RL for each sample on the same dataset with the respective 4, 5 and 6 drivers. The scatter plots show a similar performance across each driver instance.

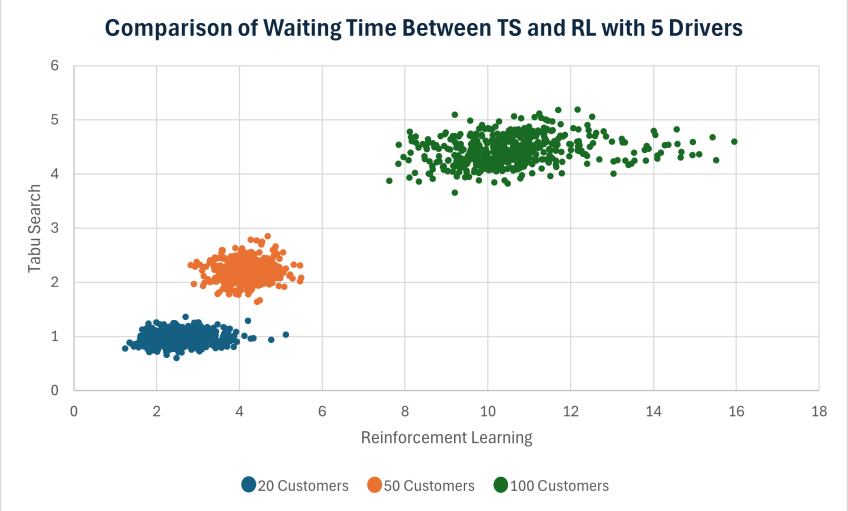
When comparing instances of 20 customers, the data points on each scatter plot show a cluster of data located close to the x-axis and y-axis, indicating low waiting times for both TS and RL. The waiting times are slightly more spread along the x-axis, suggesting a wider range in values when using RL.

When comparing scenarios with 50 customers, the instance with 4 drivers showed a wider spread of data points along the RL values compared to instances with 5 and 6 drivers, with less spread and more tightly packed data points. Across all driver instances with 50 customers, TS values were overall lower in terms of waiting time than RL.

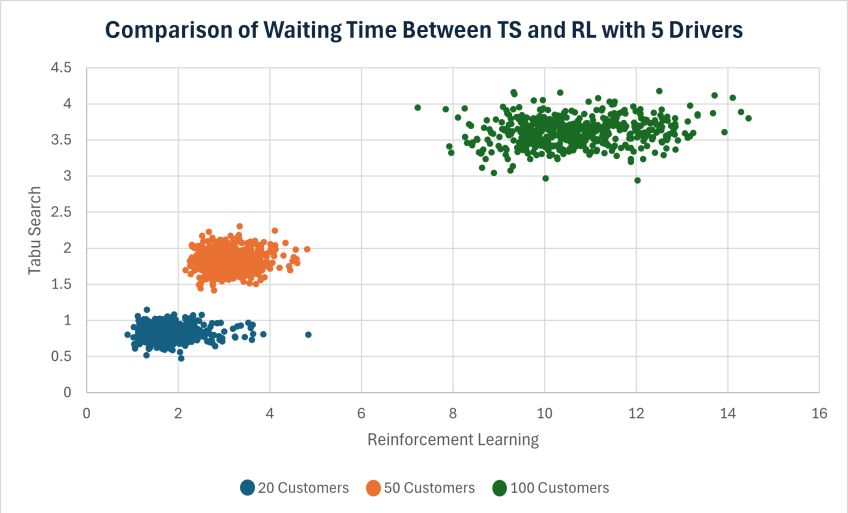
The scatter plots showed that for each driver with 100 customers, the waiting time values were spread mainly across the x-axis (RL values) and contained higher values compared to the y-axis (TS values). This showed that TS performed better with lower waiting times and less variance between samples compared to RL.



(a) 4 Drivers



(b) 5 Drivers



(c) 6 Drivers

Figure 5.26 Scatter plot comparing Waiting Time Between TS and RL

Figures 5.27 to 5.29 illustrate the average travel impact time for TS and RL across different driver instances with 20, 50, and 100 customers. The bar charts indicate that TS performed better with lower overall travel impact time values across each scenario.

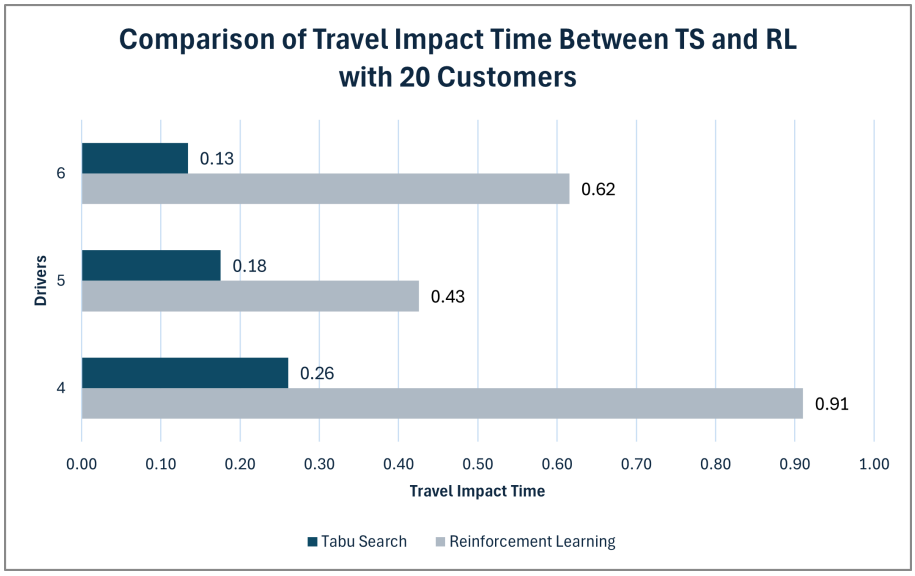


Figure 5.27 Comparing Average Travel Impact Time between TS and RL with 20 customers across each driver instance

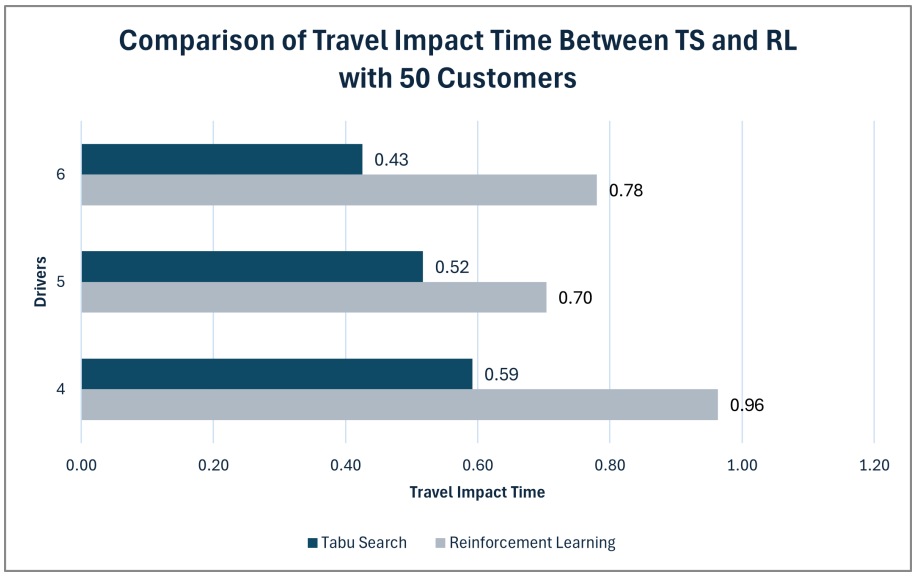


Figure 5.28 Comparing Average Travel Impact Time between TS and RL with 50 customers across each driver instance

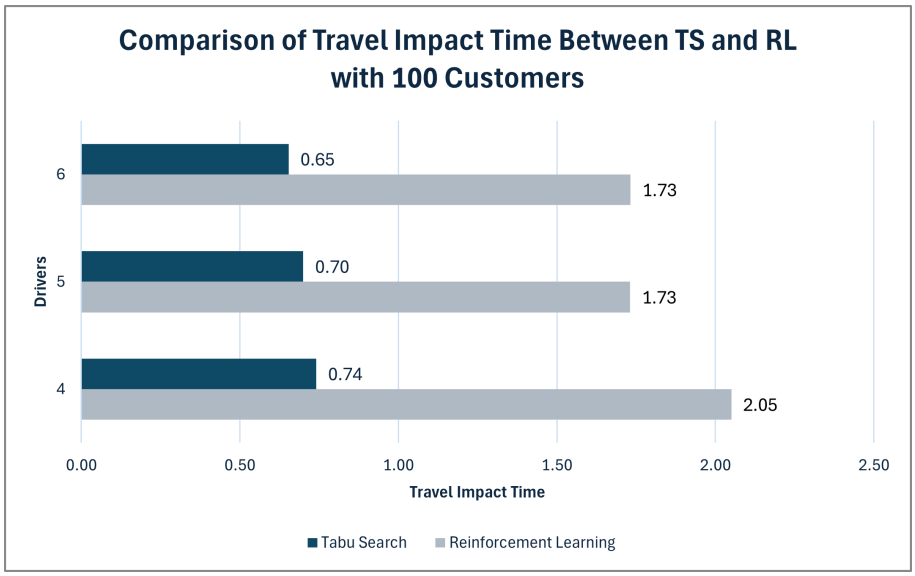
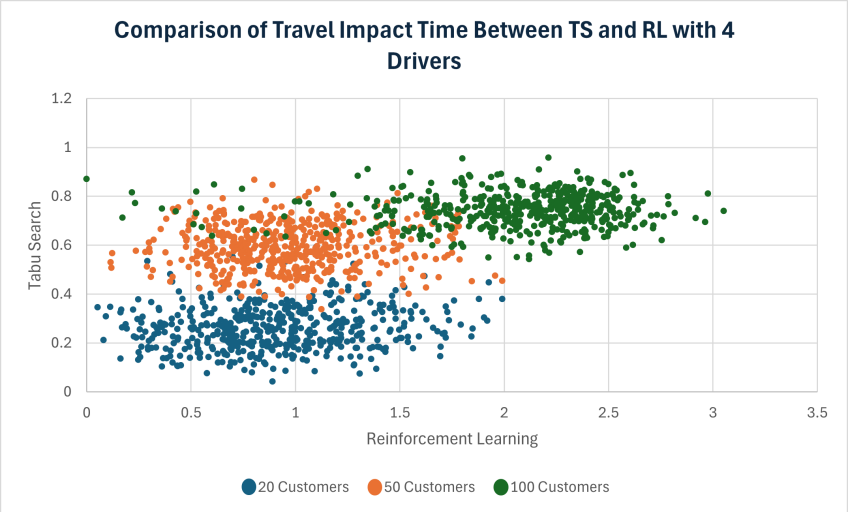


Figure 5.29 Comparing Average Travel Impact Time between TS and RL with 100 customers across each driver instance

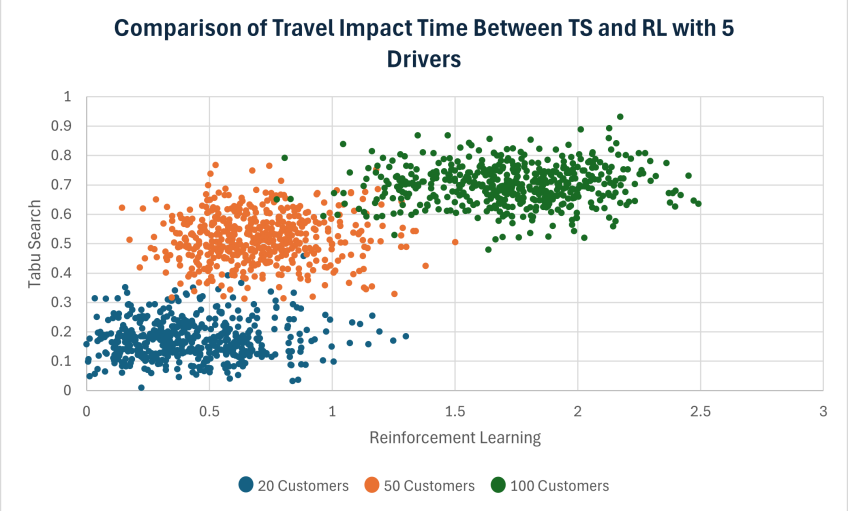
The scatter plots in Figure 5.30 display the travel impact time performance for both TS and RL experiments using different customers and drivers. Both models were compared on the same datasets.

Each group of customers showed a wide spread of data points across the x-axis (RL values) with higher values along the same axis. This indicates that the RL had a larger variation and higher travel impact times than TS.

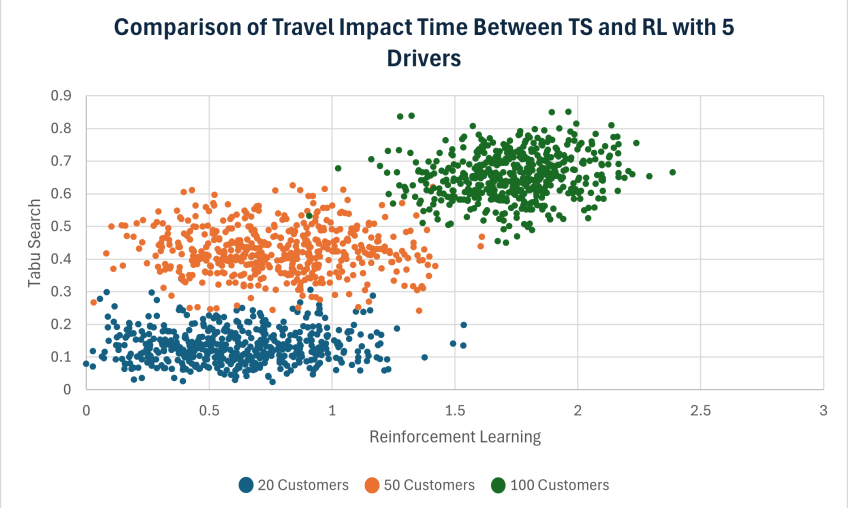
As the number of customers increased across all drivers, the data points for each customer group moved slightly further up, indicating a slight increase in travel impact time when using TS. However, the travel impact times were still significantly lower than RL. When comparing the travel impact time across multiple customer groups, it is proven that the customers spent less time between pickup and dropoff when using TS.



(a) 4 Drivers



(b) 5 Drivers



(c) 6 Drivers

Figure 5.30 Scatter Plot comparing Travel Impact Time Between TS and RL

The average drivers' distances of the TS and RL, as illustrated in Figures 5.31 to 5.33, show the performance of the two models across instances of 20, 50 and 100 customers with different numbers of drivers. The bar charts show that TS had the least distance travelled across all scenarios compared to RL, highlighting its effectiveness at optimising the routes to reduce the overall distance.

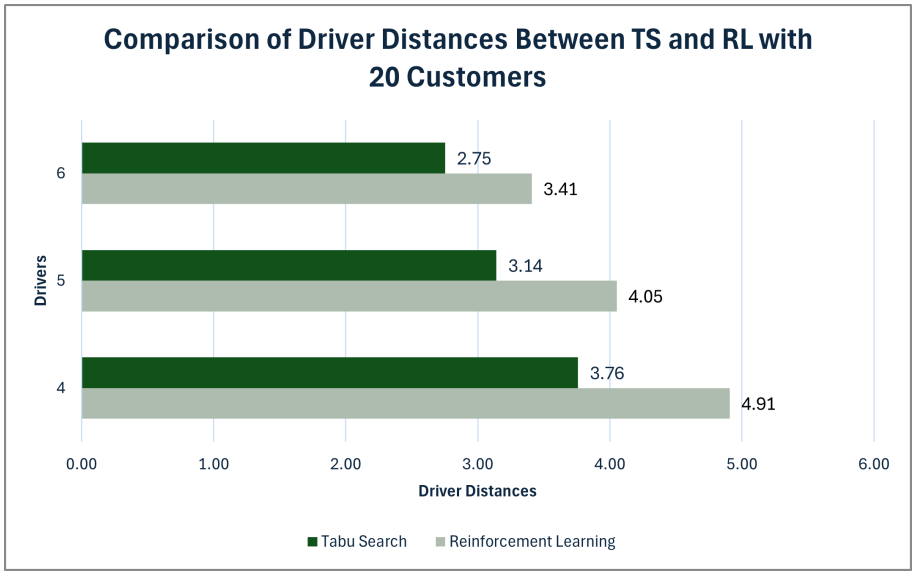


Figure 5.31 Comparing Average Distance Travelled between TS and RL with 20 customers across each driver instance

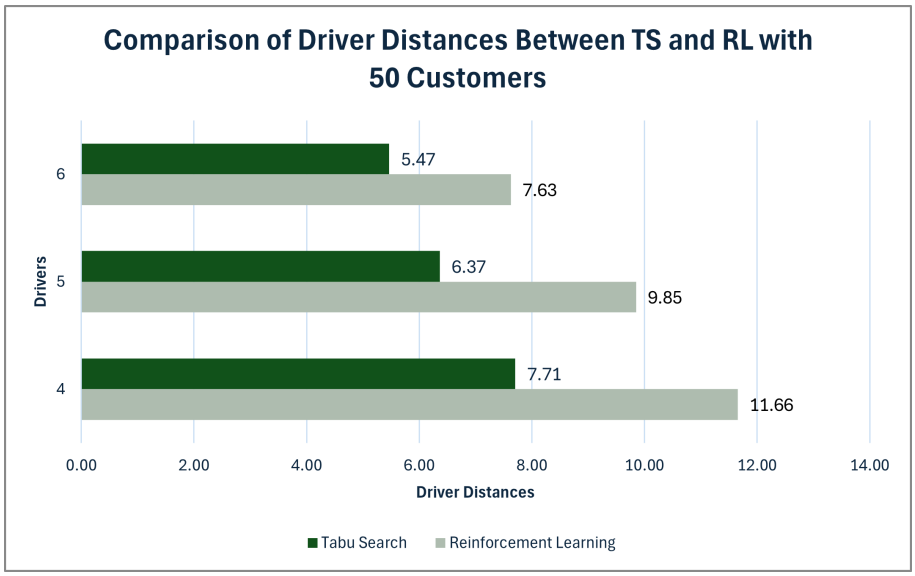


Figure 5.32 Comparing Average Distance Travelled between TS and RL with 50 customers across each driver instance

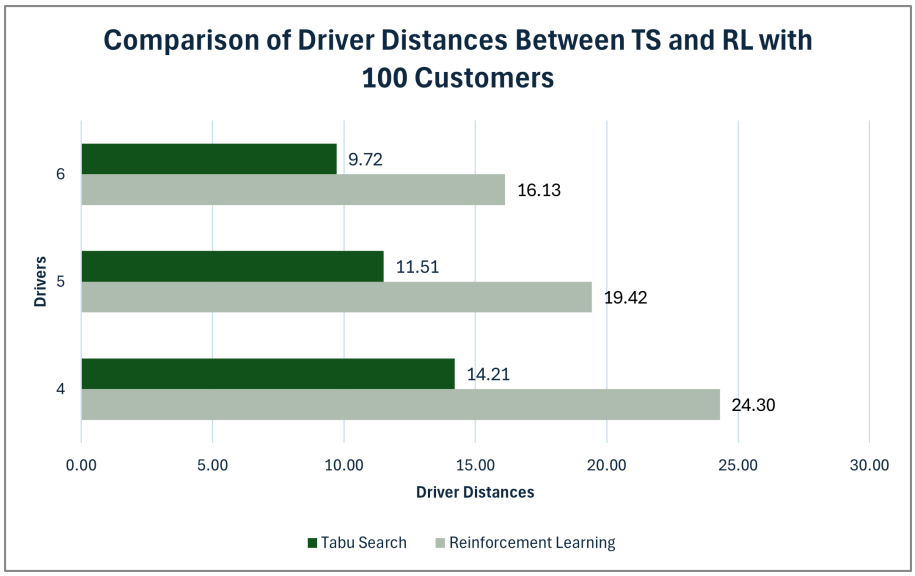
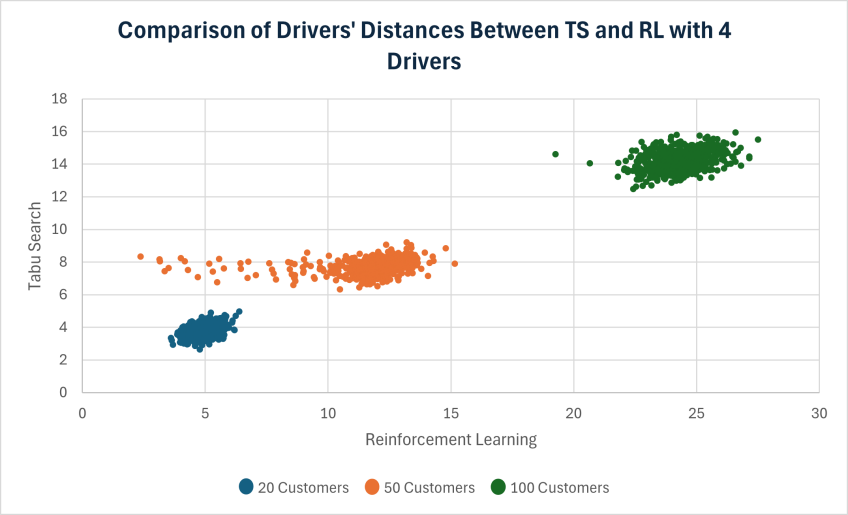


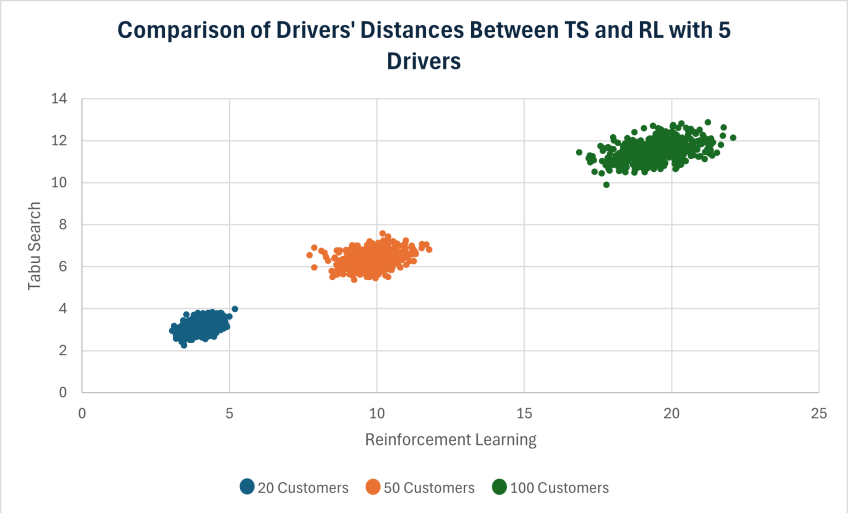
Figure 5.33 Comparing Average Distance Travelled between TS and RL with 100 customers across each driver instance

Figure 5.34 shows the performance of the TS and RL when comparing the total distance travelled on each sample across instances of 4, 5 and 6 drivers with multiple groups of customers. The scatter plots show clusters of data points for each customer group across all drivers. The clusters indicate that as the number of customers increased, both models saw an increase in distance travelled. The scatter plots also show that as the number of customers in an instance increases, the margin in the distance travelled values between the two models increases, with RL having higher distances than TS. The scatter plots also showed a slight increase in the spread of data points as the number of customers increased. In the instance of 50 customers with 4 drivers, an anomaly increased the number of outliers along the RL x-axis, causing an increase in the variation of distance travelled.

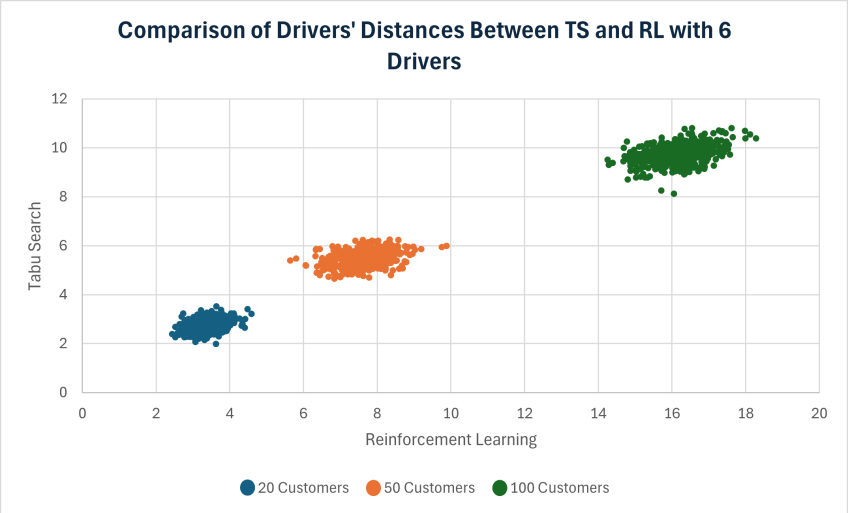
It is evident that using TS to solve the solution can reduce the total distance travelled by the vehicles across multiple customers.



(a) 4 Drivers



(b) 5 Drivers



(c) 6 Drivers

Figure 5.34 Scatter plot comparing Drivers' Distances Between TS and RL

A comparison of the average running times for both TS and RL is illustrated in Figures 5.35 to 5.37 for instances consisting of 20, 50 and 100 customers. The results show that TS was faster when solving smaller problems, as shown in instances with 20 customers, but struggled with larger problems, as shown in instances with 50 and 100 customers. As the number of customers increased, the TS computation time increased exponentially due to the increased complexity and size of the search space, requiring it to spend more time exploring and evaluating each candidate solution at each iteration.

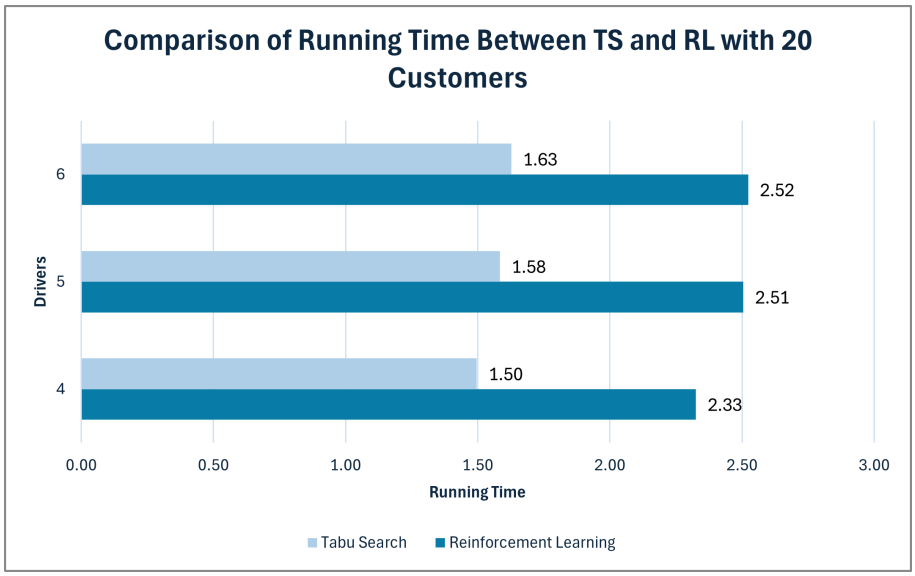


Figure 5.35 Comparing Average Running Time between TS and RL with 20 customers across each driver instance

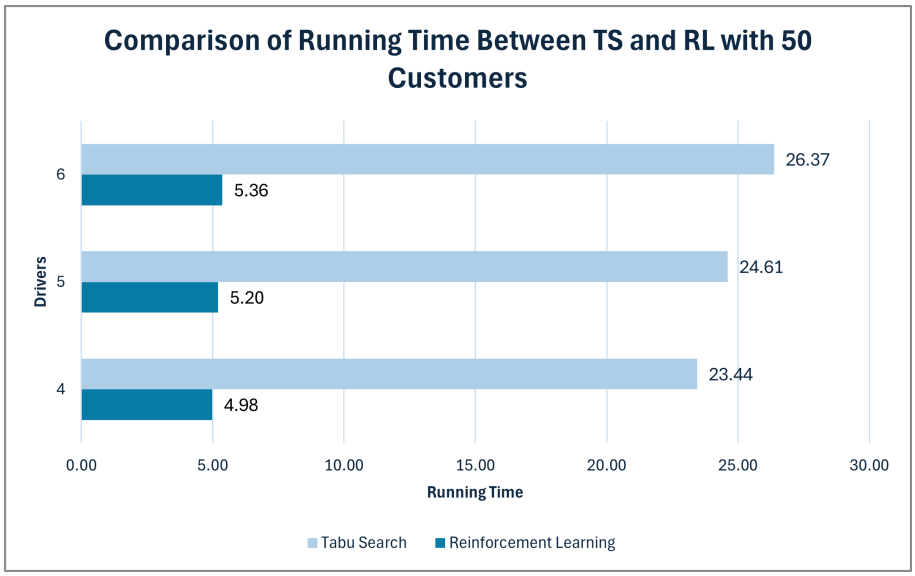


Figure 5.36 Comparing Average Running Time between TS and RL with 50 customers across each driver instance

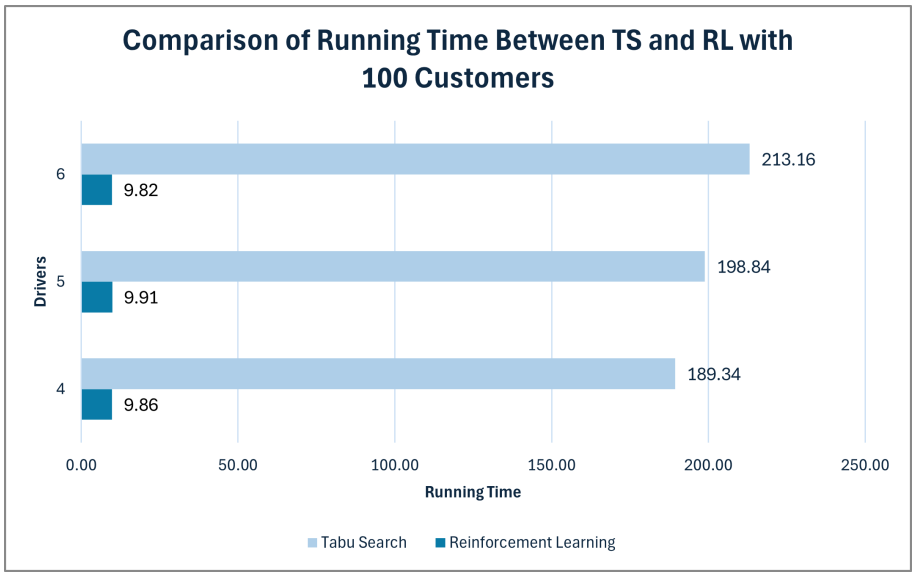


Figure 5.37 Comparing Average Running Time between TS and RL with 100 customers across each driver instance

Scatter plots were used to compare the running time performance of TS and RL when using the same dataset. Figures 5.38 to 5.40 illustrate the running time performance across instances with 4, 5 and 6 drivers, respectively. Each data point represents a sample from the dataset, indicating its performance when using RL (x-axis) and TS (y-axis).

The scatter plots show that across each driver instance, the points belonging to 20 customers form a nearly straight line with a narrow height across the x-axis. This indicates that in instances with 20 customers, the RL running time had a higher range of running time values than that of TS. When comparing the data points belonging to 50 customers, there is a cluster of data where the values on the y-axis are larger than those on the x-axis, suggesting that TS overall required more computational time than RL. The data points belonging to 100 customers for each driver instance are spread across the y-axis while remaining within a narrow range on the x-axis, showing a considerable variation in running time when using TS. The TS values on the y-axis were also considerably higher than those of RL on the x-axis, indicating a large increase in computation time compared to RL. This confirms that RL outperformed TS in terms of execution time on larger instances, highlighting RL's ability to handle large-scale MVRRPP instances by learning optimal policies that generalise different problem sizes.

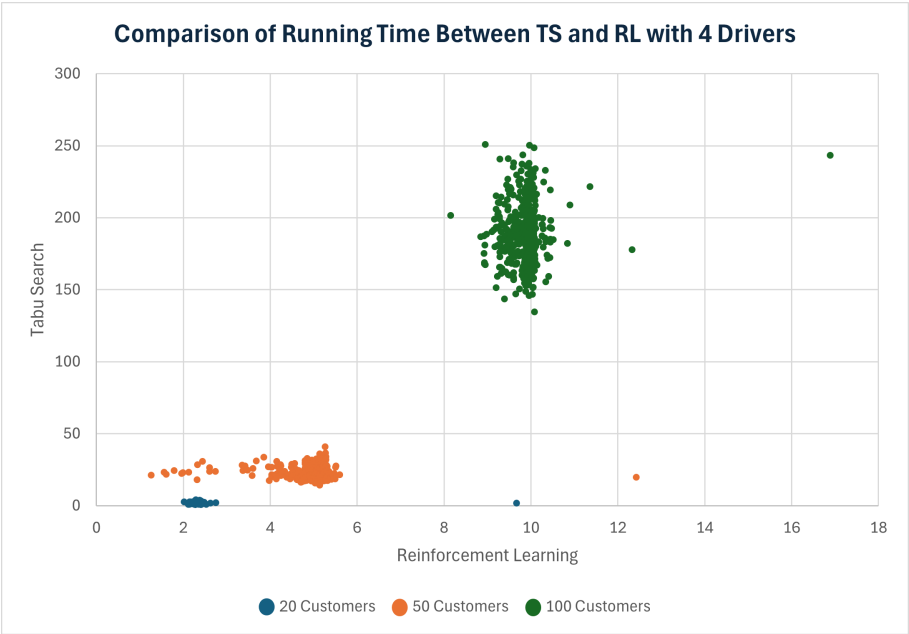


Figure 5.38 Scatter Plot comparing the Running Time between TS and RL with 4 drivers across multiple customers

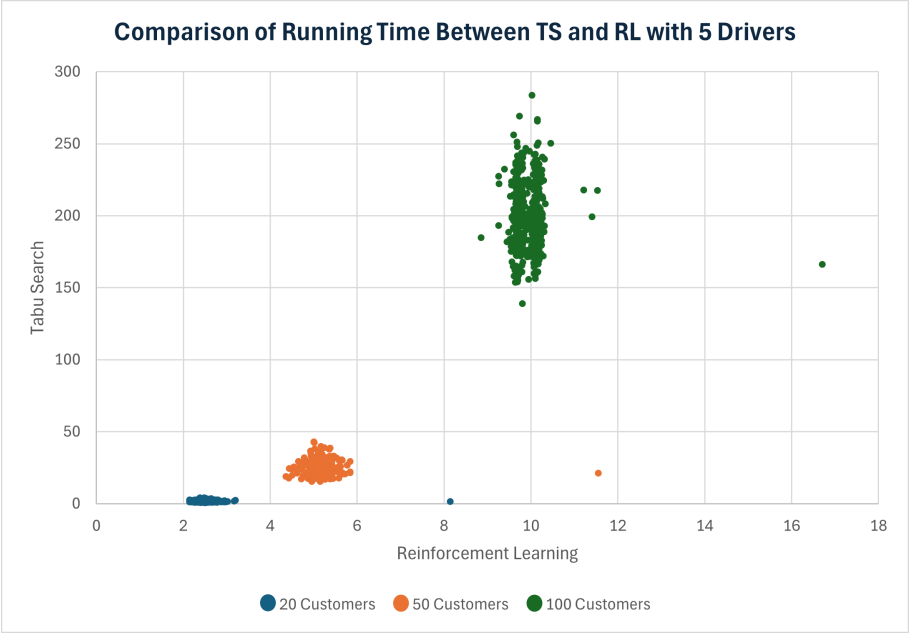


Figure 5.39 Scatter Plot comparing the Running Time between TS and RL with 5 drivers across multiple customers

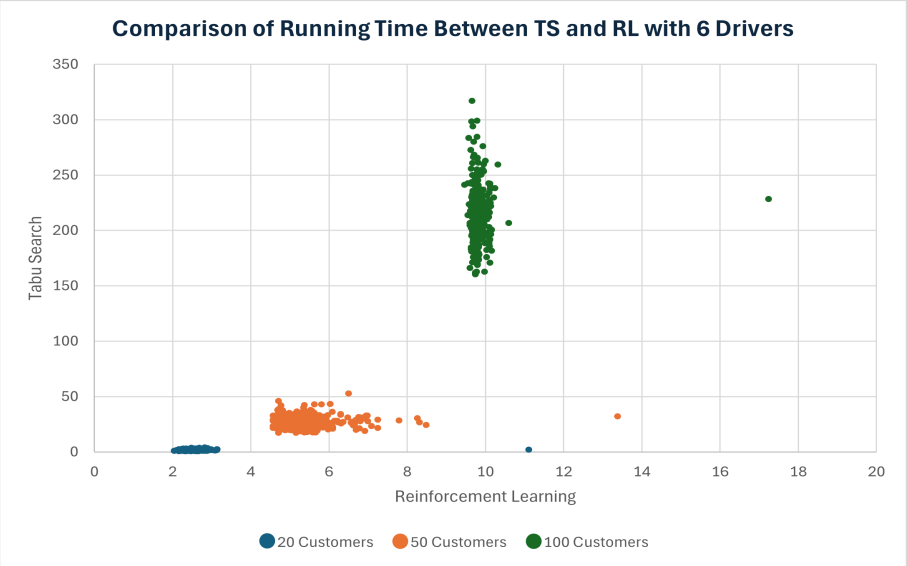


Figure 5.40 Scatter Plot comparing the Running Time between TS and RL with 6 drivers across multiple customers

5.3 Objective 3 - Initialising Tabu Search with a solution generated using Reinforcement Learning

The third experiment involved using the solution obtained from RL in Experiment 2 as the initial solution for TS. The performance was then evaluated and compared to the previous objectives using the respective datasets of each scenario.

Figure 5.41 illustrates a graph instance of 20 customers and 4 drivers while Figure 5.42 illustrates the final solution achieved when using TS with RL on this instance. A graph with each driver's route from the final solution is depicted in Figure 5.43.

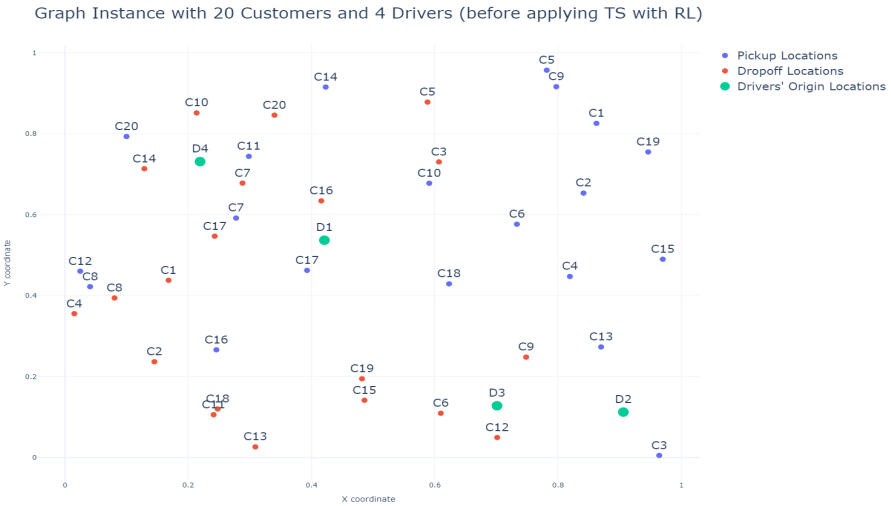


Figure 5.41 Graph Instance with 20 Customers and 4 Drivers before using TS with RL

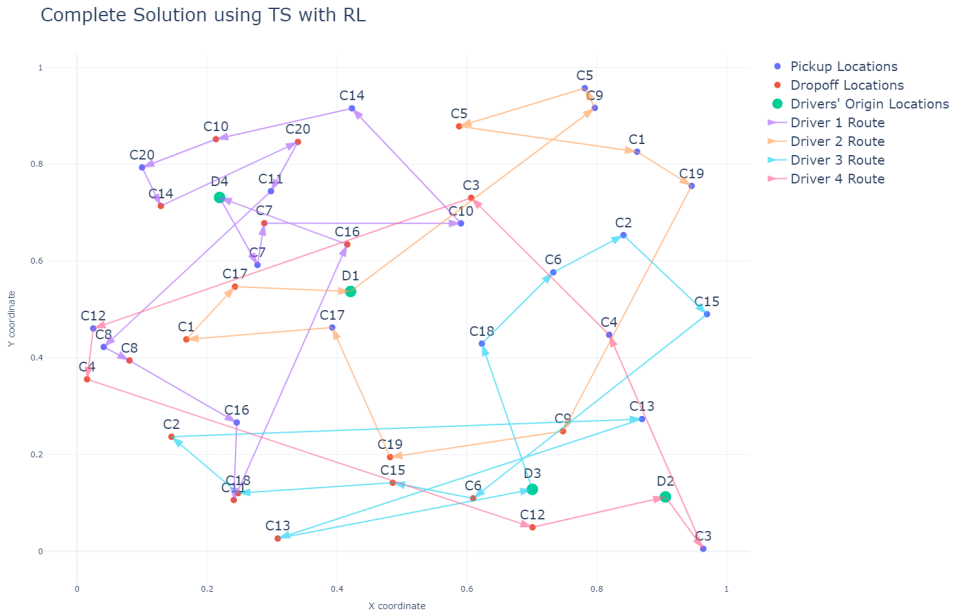


Figure 5.42 Final Solution after applying TS with RL on the graph instance of 20 customers and 4 drivers

(a) Driver 1 Route

(b) Driver 2 Route

(c) Driver 3 Route

(d) Driver 4 Route

Figure 5.43 Drivers' Routes obtained using TS with RL on an instance with 4 drivers and 20 customers

Figure 5.44 illustrates the waiting time performance when using TS with RL with instances of different drivers and customers. The box plot shows that for each driver instance, the waiting time increased as more customers were added. The box plots also showed that the overall waiting time decreased as more drivers were added to an instance with the same number of customers.

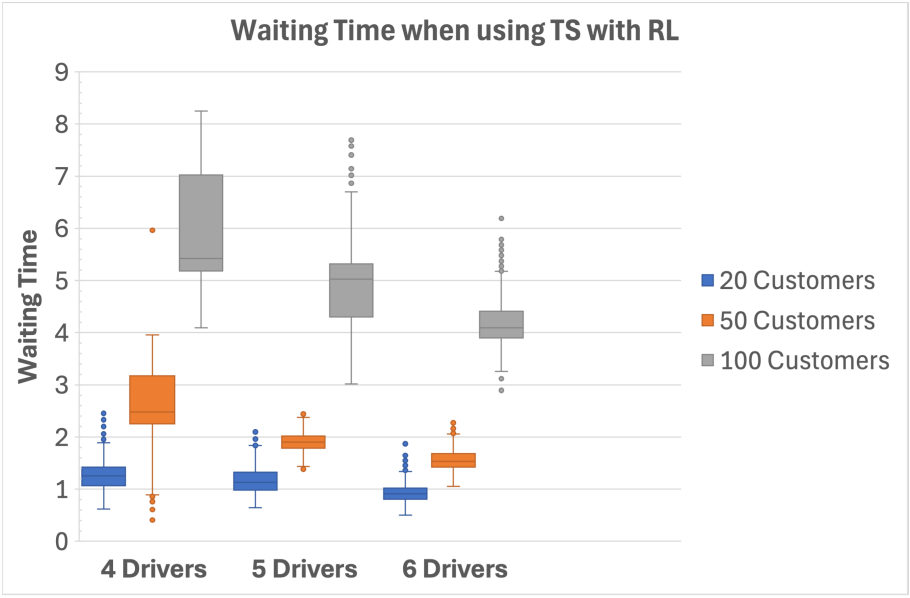


Figure 5.44 Box Plot with the Waiting Time across different instances of drivers and customers using TS with RL

As illustrated in Figure 5.45, the travel impact time increased with the addition of customers, as the drivers were required to make more stops. In instances with 20 and 50 customers, the travel impact time decreased as more drivers were available, indicating that the customers were being distributed along the routes. However, in instances with 100 customers, the travel impact time remained consistent and varied slightly across instances with 4, 5 and 6 drivers.

The box plot in Figure 5.46 illustrates the distance travelled across different instances. The box plot shows that the distance travelled by each group of drivers increased when more customers were added. This was expected since the driver needed to make additional stops for each customer's pickup and dropoff location, increasing the overall distance travelled. As evident in the box plot, the increase in drivers helped distribute customers along different routes.

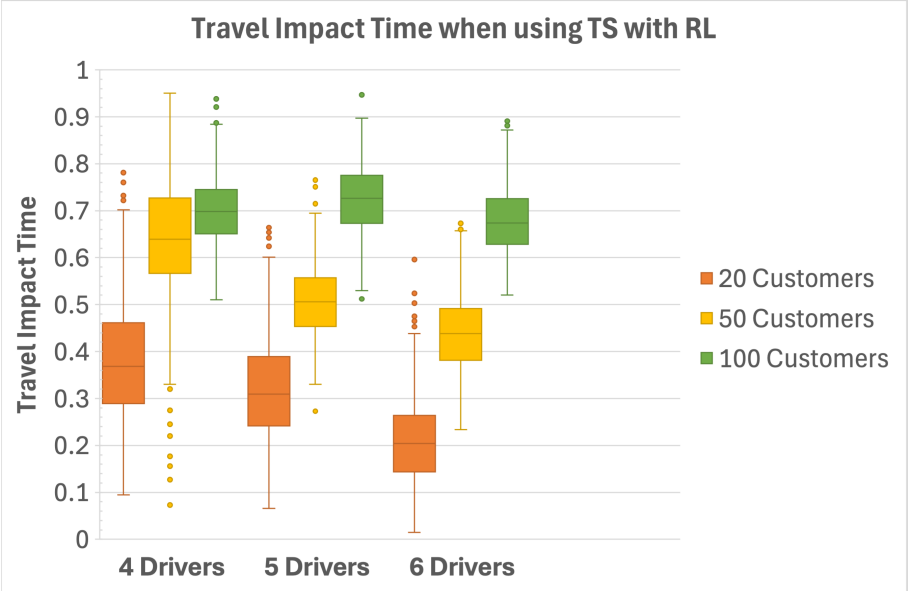


Figure 5.45 Box Plot with the Travel Impact Time across different instances of drivers and customers using TS with RL

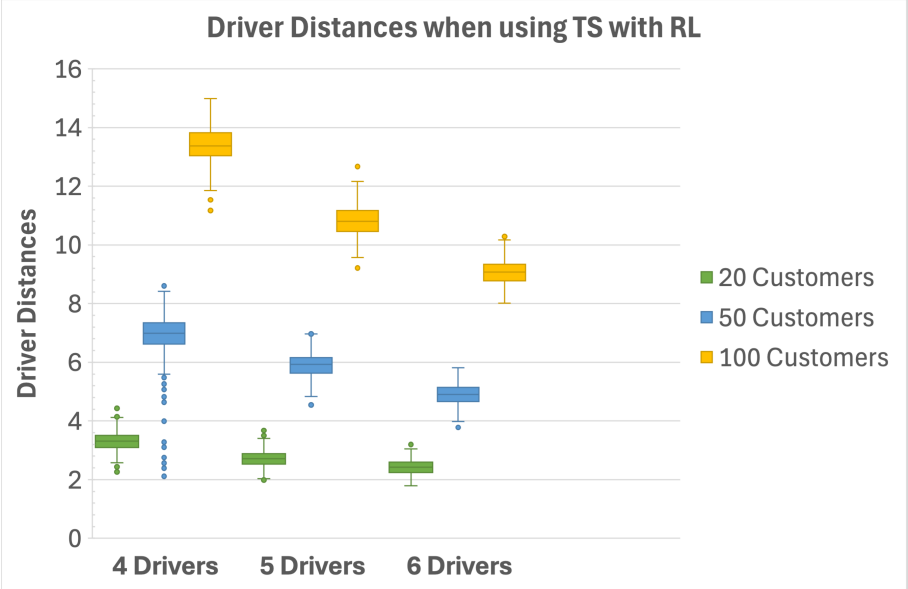


Figure 5.46 Box Plot with the Drivers' Distances across different instances of drivers and customers using TS with RL

Figure 5.47 shows the number of iterations taken by the TS with RL model to find the optimal solution. More iterations were required to find the optimal solution in instances with larger customers, for each driver group. However, the number of iterations remained consistent across each driver group with the same number of customers.

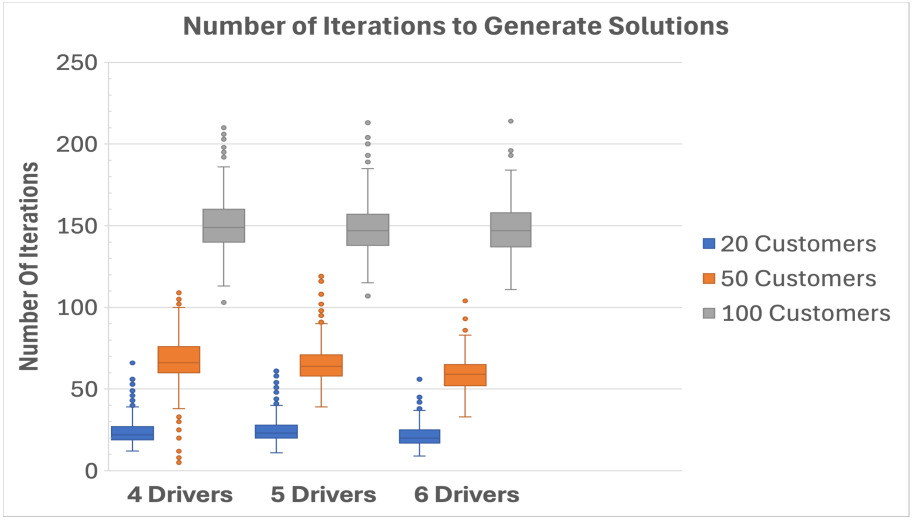
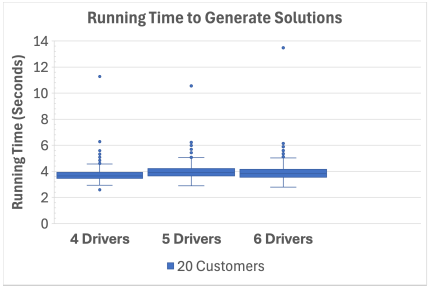
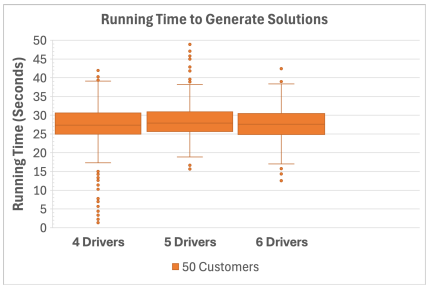


Figure 5.47 Box Plot with the number of iterations when using TS with RL

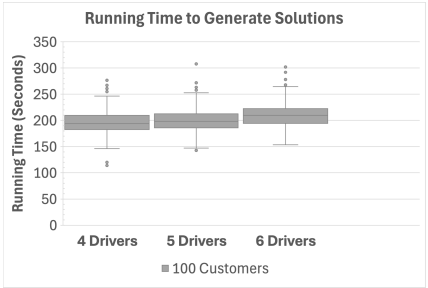
Figure 5.48 shows the running time taken by the TS with RL to generate the best solutions. The box plots show that the running time experienced exponential growth when adding customers, suggesting that the TS with RL model required more computation time when finding the optimal solution in the larger search space.



(a) 20 Customers



(b) 50 Customers



(c) 100 Customers

Figure 5.48 Running Time to generate solutions using TS

Figure 5.49 to 5.51 illustrate bar charts representing the average waiting time performance of TS using RL compared with the waiting time of TS (Objective 1) and RL (Objective 2) across 20, 50 and 100 customers, respectively. The lower the bar, the better the performance since the aim is to minimise each metric. The bar charts show a similar waiting time performance between TS and TS with RL. Instances with customers of 20 (Figure 5.49) and 100 (Figure 5.51) indicate that TS performed best in terms of waiting time, followed by TS with RL and finally RL. In instances with 50 customers (Figure 5.50), the TS with RL outperformed TS while RL followed last.

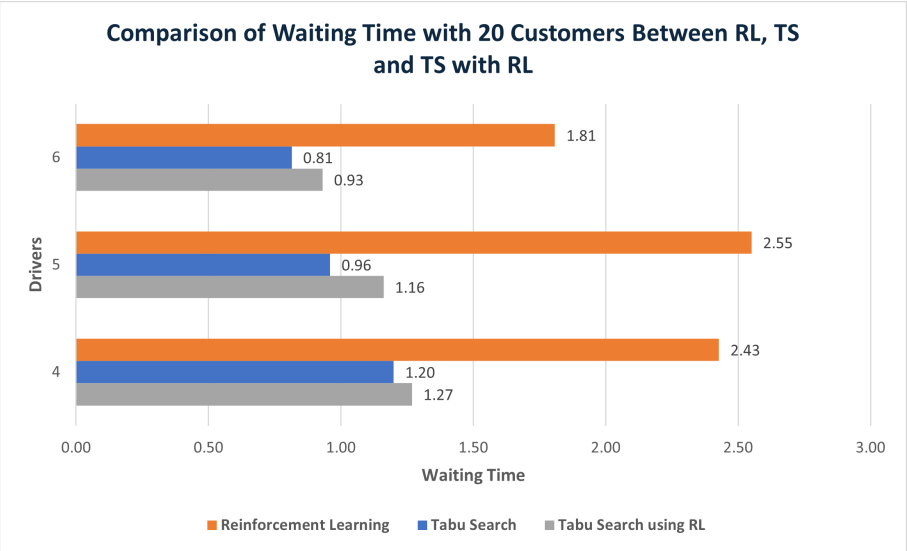


Figure 5.49 Comparing Average Waiting Time with 20 customers across each driver instance between RL, TS and TS with RL

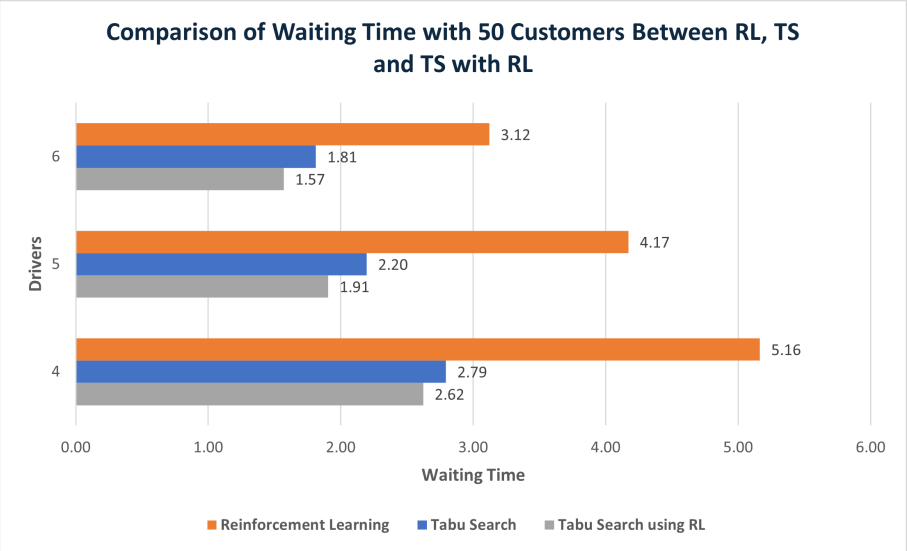


Figure 5.50 Comparing Average Waiting Time with 50 customers across each driver instance between RL, TS and TS with RL

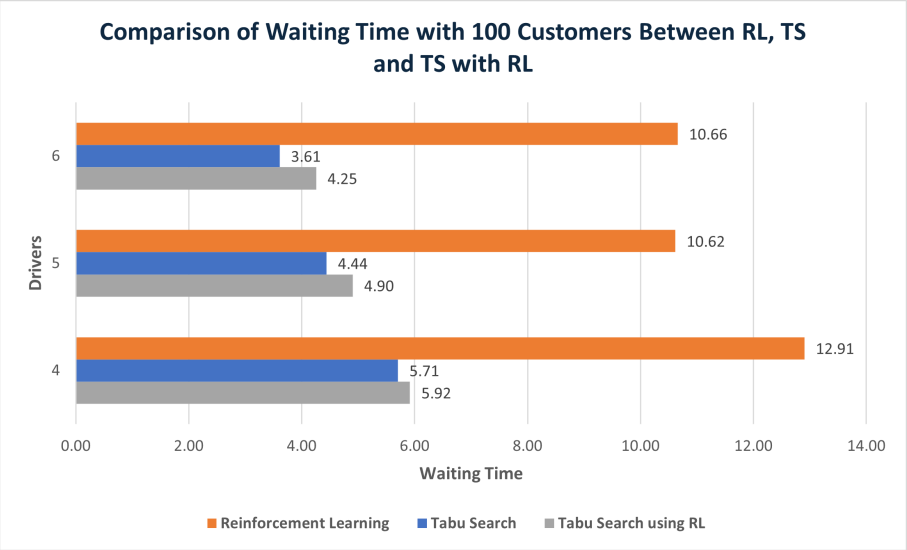
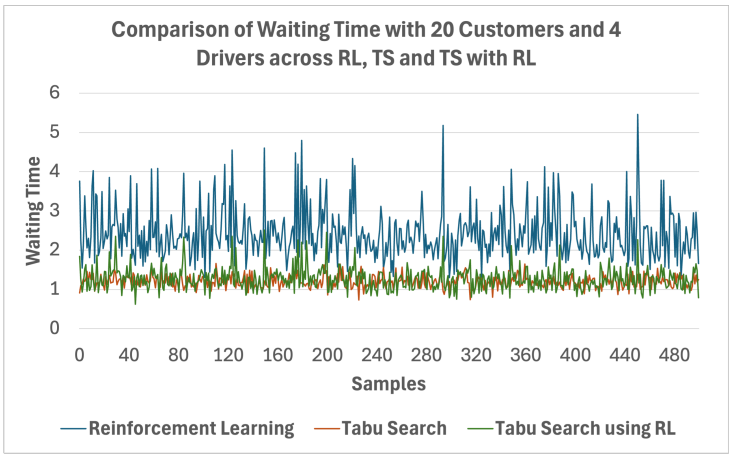


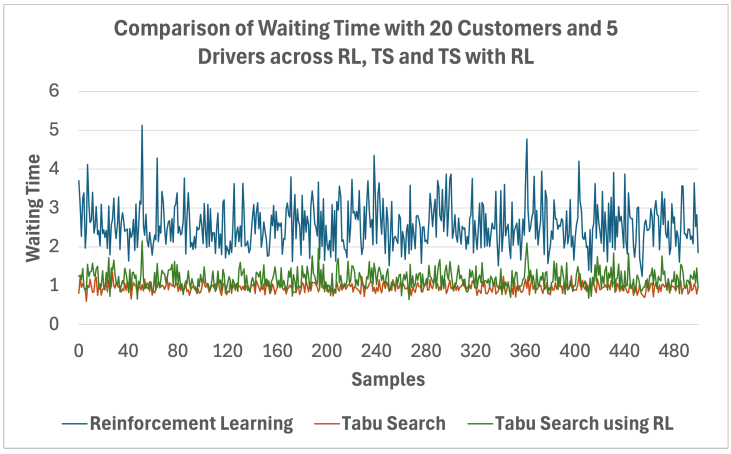
Figure 5.51 Comparing Average Waiting Time with 100 customers across each driver instance between RL, TS and TS with RL

Figures 5.52 to 5.54 are line charts representing the waiting times across all experiments for each sample using the same dataset. Each line chart represents the waiting time with 20, 50, and 100 customers across 4, 5 and 6 drivers. All line charts show fluctuations in waiting time across different models, highlighting how each problem varies in complexity with each sample.

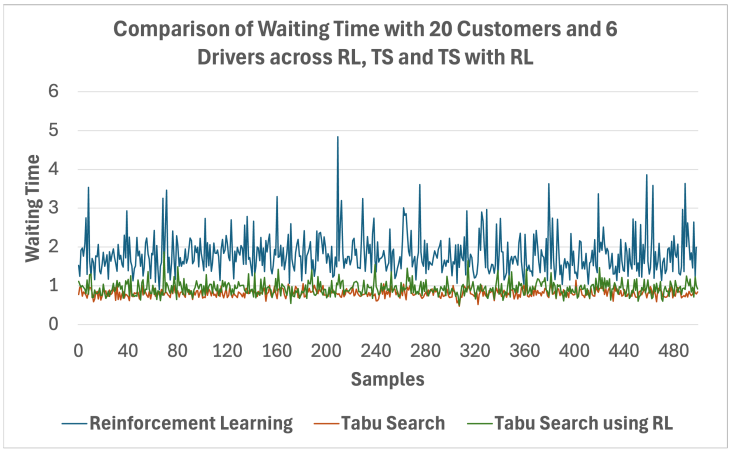
Figure 5.52 shows the waiting time for each model across 20 customers. The line charts shows that the performance of TS and TS with RL across each driver instance was relatively similar, with TS with RL showing a slightly higher range in values. Figure 5.53 shows the waiting time across 50 customers for each model. These line charts show that TS with RL had the lowest overall range in average waiting time values when compared to TS and RL. The line representing RL showed an overall decrease in values between 4 and 5 drivers, indicating a performance improvement, with 6 drivers also remaining similar in terms of waiting time performance to that of 5 drivers. Figure 5.54 shows the waiting time performance across 100 customers. In these line charts, TS had the lowest range in waiting time values, with TS with RL having slightly higher values. RL had the highest waiting time values, which however decreased as the number of drivers increased.



(a) 4 Drivers

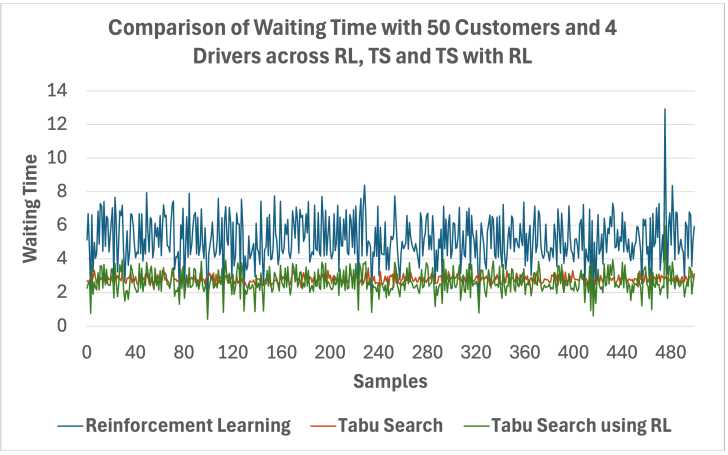


(b) 5 Drivers

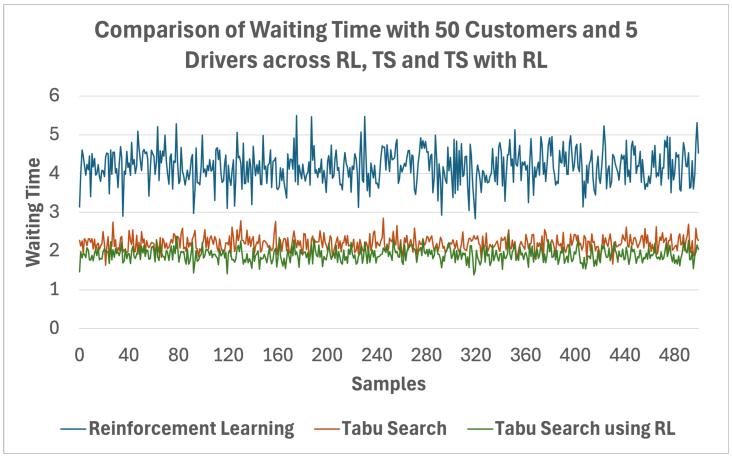


(c) 6 Drivers

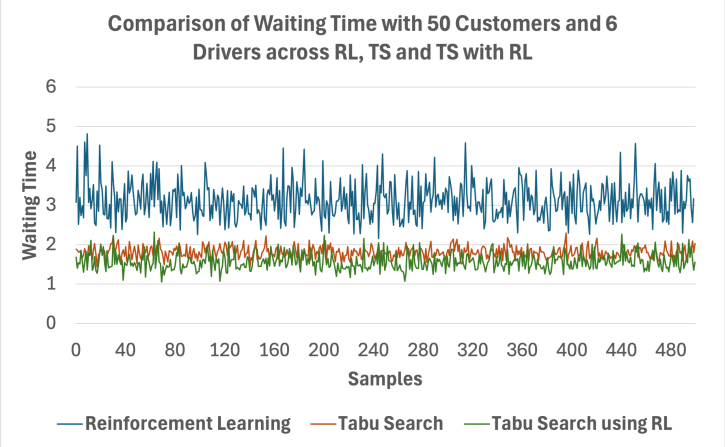
Figure 5.52 Line Chart comparing Waiting Time between RL, TS and TS with RL across samples with 20 customers



(a) 4 Drivers

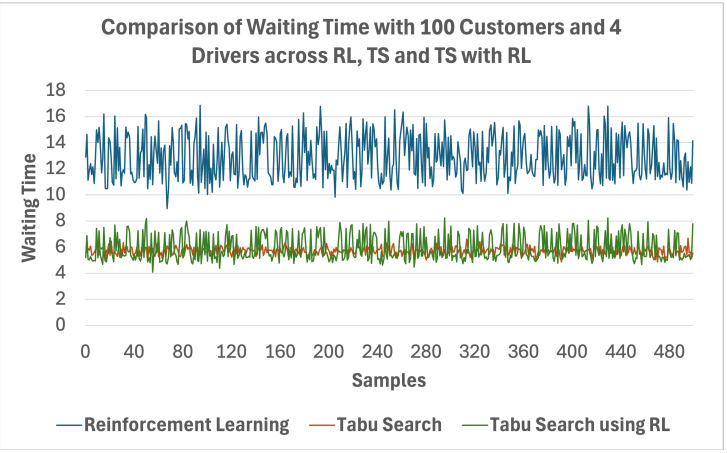


(b) 5 Drivers

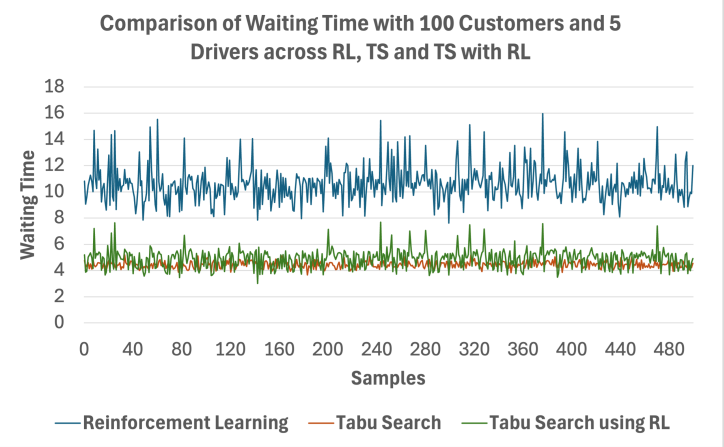


(c) 6 Drivers

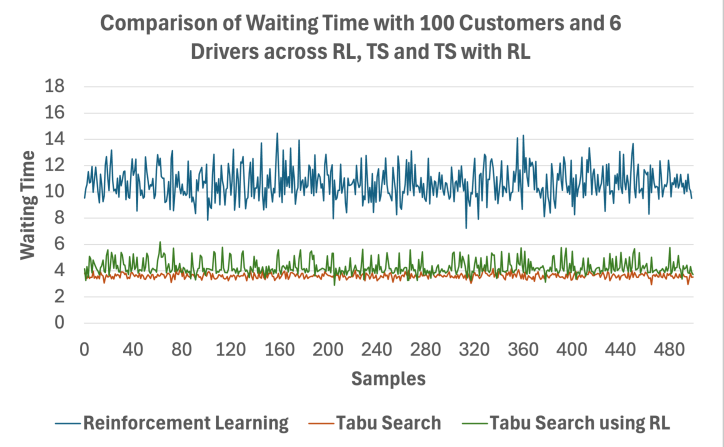
Figure 5.53 Line Chart comparing Waiting Time between RL, TS and TS with RL across samples with 50 customers



(a) 4 Drivers



(b) 5 Drivers



(c) 6 Drivers

Figure 5.54 Line Chart comparing Waiting Time between RL, TS and TS with RL across samples with 100 customers

Figures 5.55 to 5.57 display the average travel impact time for the three experiments when evaluated on 20, 50 and 100 customers across different drivers. As observed in Figure 5.55, TS achieved the lowest travel impact time average, followed by TS with RL, and RL with the highest average. On the other hand, Figure 5.56 and Figure 5.57 reveal a similar performance between TS and TS with RL, with, at times, TS with RL surpassing TS. Overall, the average travel impact time increased with the addition of customers across all experiments. In all figures examined, RL consistently exhibited the highest time customers spent from pickup to dropoff compared to the TS models. This observation highlights the challenges faced by RL in optimisation vehicle routing solutions effectively.

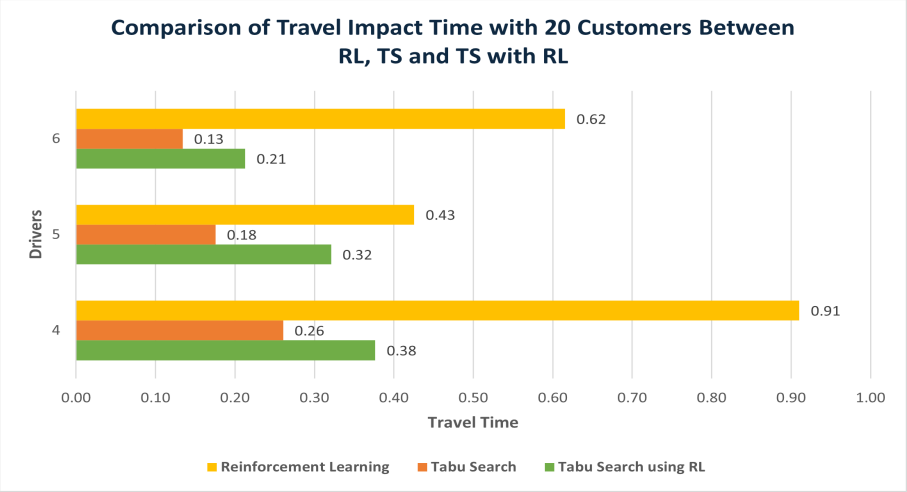


Figure 5.55 Comparing Average Travel Impact Time with 20 customers across each driver instance between RL, TS and TS with RL

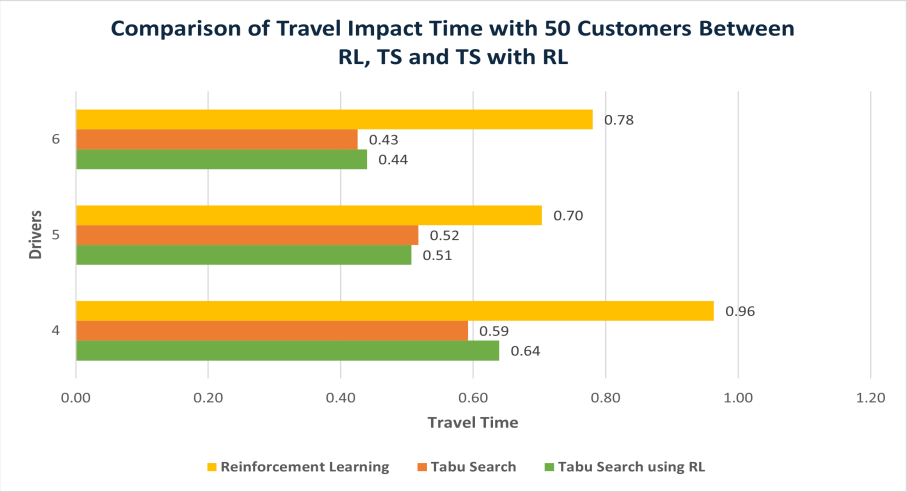


Figure 5.56 Comparing Average Travel Impact Time with 50 customers across each driver instance between RL, TS and TS with RL

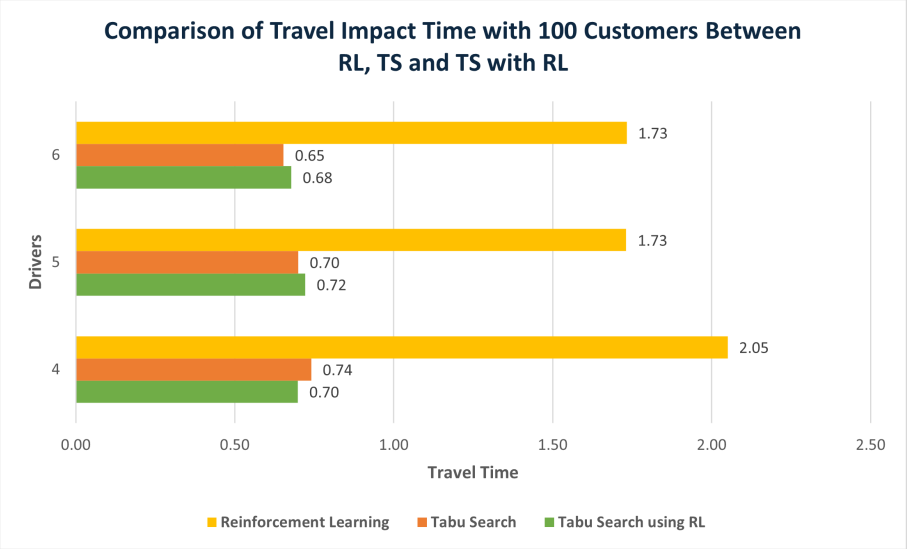
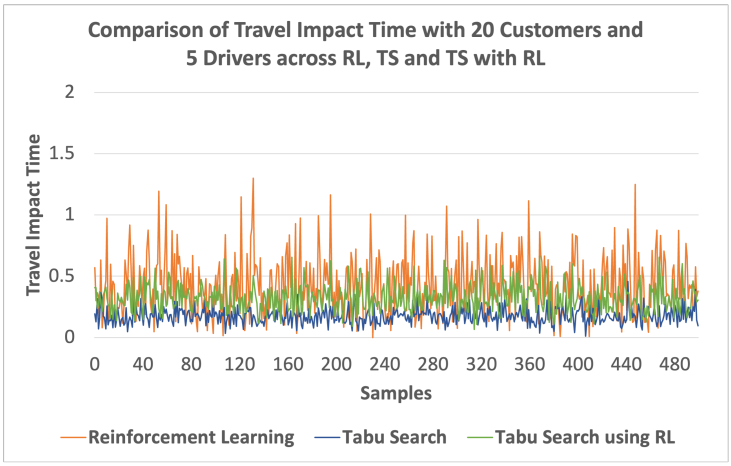


Figure 5.57 Comparing Average Travel Impact Time with 100 customers across each driver instance between RL, TS and TS with RL

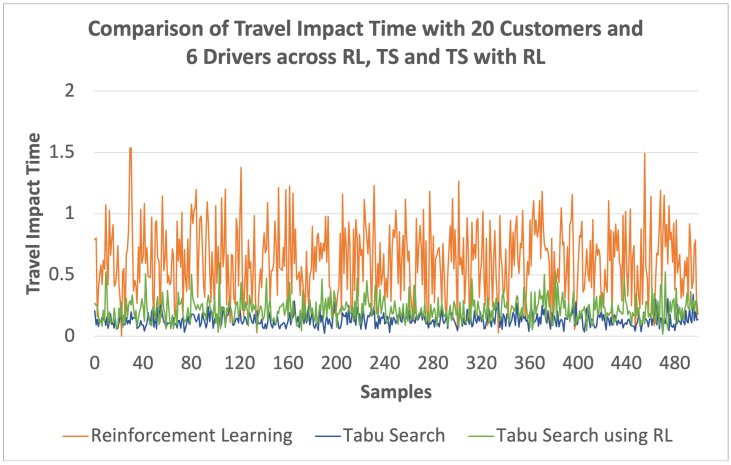
Figures 5.58 to 5.60 illustrate line charts that depict the travel impact time performance for each sample in the dataset for the three experiments. Across all figures, the line depicting RL's performance showed large spikes throughout, indicating substantial variations in the travel impact time across different instances. The variations suggest that RL faced challenges in minimising customers' time between pickup and dropoff while optimising routing efficiency.

In Figure 5.58, TS showed the best performance, achieving the lowest range and travel impact time values across each group of drivers with 20 customers, whilst TS with RL had slightly higher values and range followed by RL. Figure 5.59 and Figure 5.60 showed similar performance between TS and TS with RL across all groups, excluding the instance with 100 customers and 6 drivers, where the performance was slightly better when using TS with a lower range in values.

(a) 4 Drivers

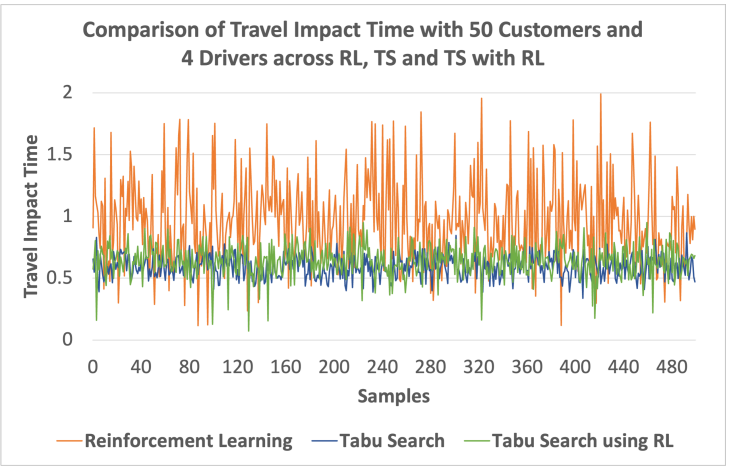


(b) 5 Drivers

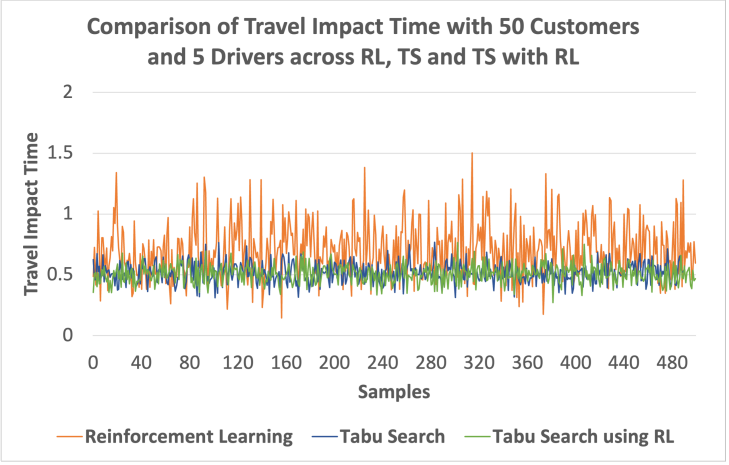


(c) 6 Drivers

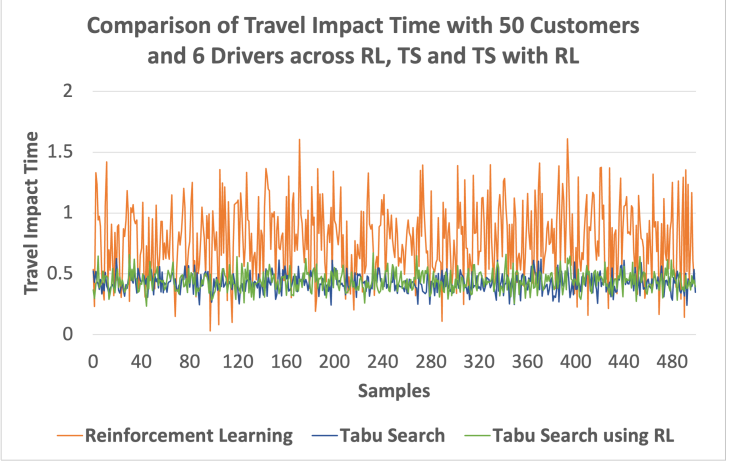
Figure 5.58 Line Chart comparing Travel Impact Time between RL, TS and TS with RL across samples with 20 customers



(a) 4 Drivers

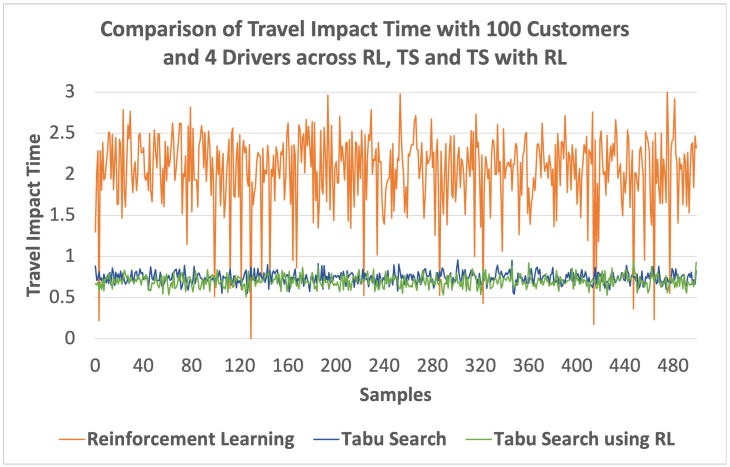


(b) 5 Drivers

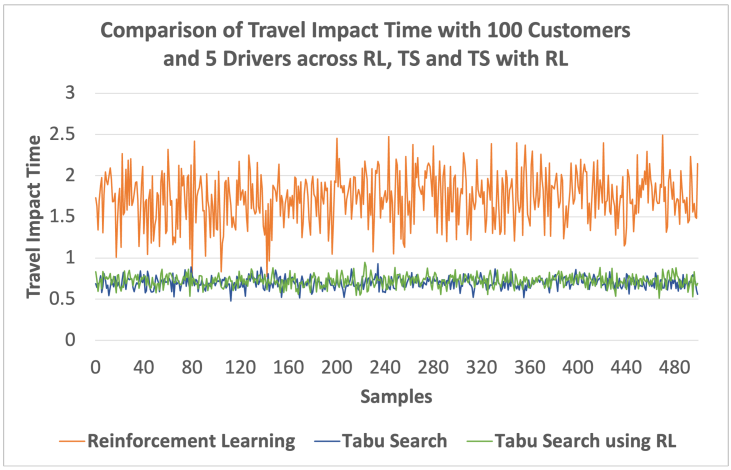


(c) 6 Drivers

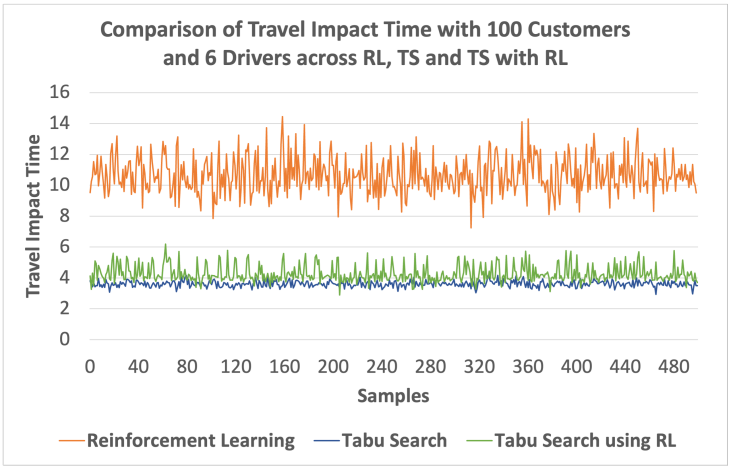
Figure 5.59 Line Chart comparing Travel Impact Time between RL, TS and TS with RL across samples with 50 customers



(a) 4 Drivers



(b) 5 Drivers



(c) 6 Drivers

Figure 5.60 Line Chart comparing Travel Impact Time between RL, TS and TS with RL across samples with 100 customers

The bar charts in Figures 5.61 to 5.63 depict the total average distance travelled when evaluating the three experiments on groups of 20, 50 and 100 customers with different drivers. The bar charts indicate that TS with RL performed best with the least distance travelled across each instance with different numbers of customers and driver groups. The total distance travelled was slightly higher when using TS but still considerably lower than RL's. As expected, the bar charts indicate that as the number of drivers increases, the distance travelled decreases across all models with the same number of customers. Nevertheless, the total average distance travelled increases across all models as the number of customers increases.

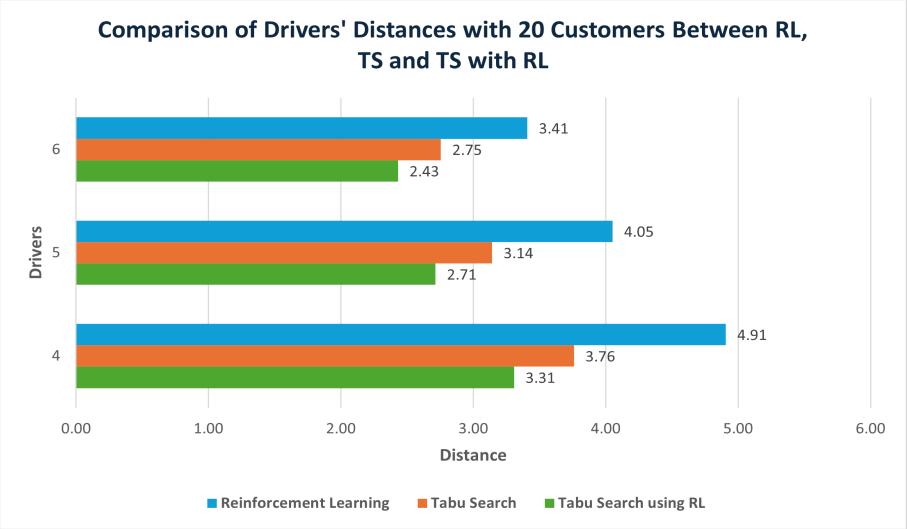


Figure 5.61 Comparing Average Drivers' Distances with 20 customers across each driver instance between RL, TS and TS with RL

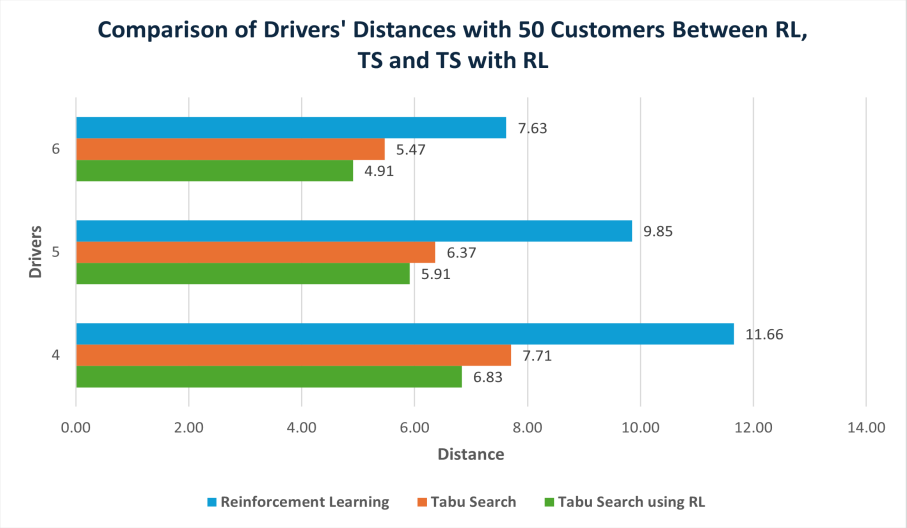


Figure 5.62 Comparing Average Drivers' Distances with 50 customers across each driver instance between RL, TS and TS with RL

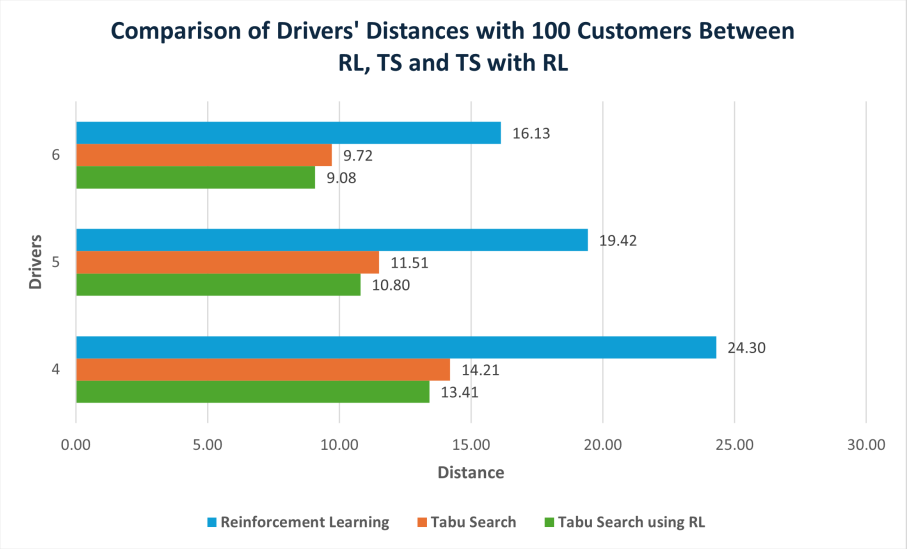
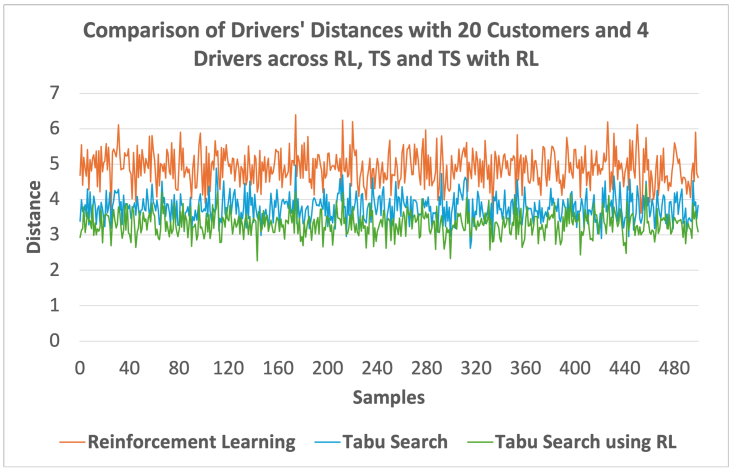
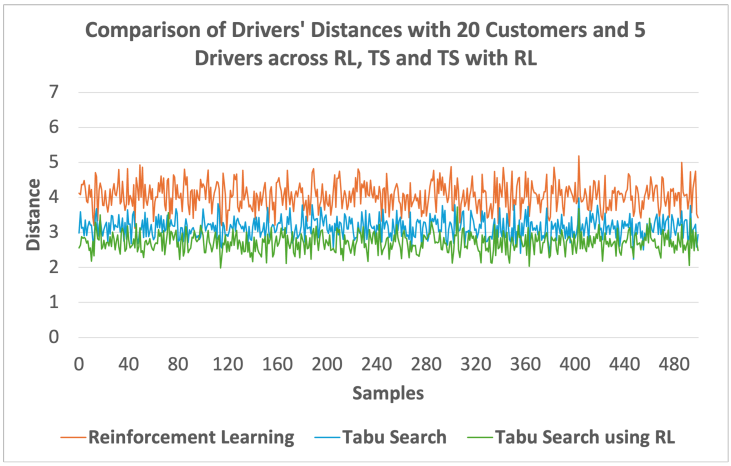


Figure 5.63 Comparing Average Drivers' Distances with 100 customers across each driver instance between RL, TS and TS with RL

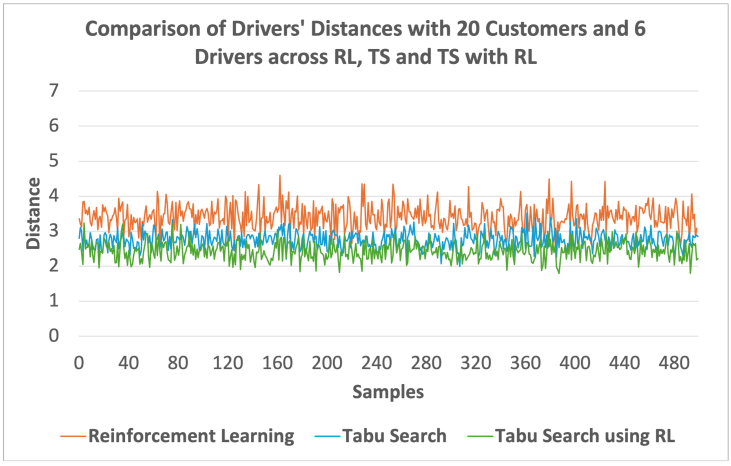
The line charts in Figures 5.64 to 5.66 depict the total distance travelled achieved for each sample in the dataset across the three experiments. The line charts indicate that the line depicting the performance of TS with RL had the lowest set of values across each group of drivers and customers, followed by TS, which had slightly higher distance values, and RL, which had the highest values. The line charts with 50 customers and 4 drivers showed high variations in the total distance travelled when using RL and TS with RL. This suggests that the solution obtained using RL may not have been optimal in terms of distance, hindering the TS with RL's ability to find a better solution, leading to spikes in the distance metric as it attempts to navigate and refine the suboptimal solution. The line charts also show a wider gap between RL's performance and other models as customer numbers increased. This suggests that the RL model may have faced challenges in scaling to larger problem instances with the increase in customers.



(a) 4 Drivers

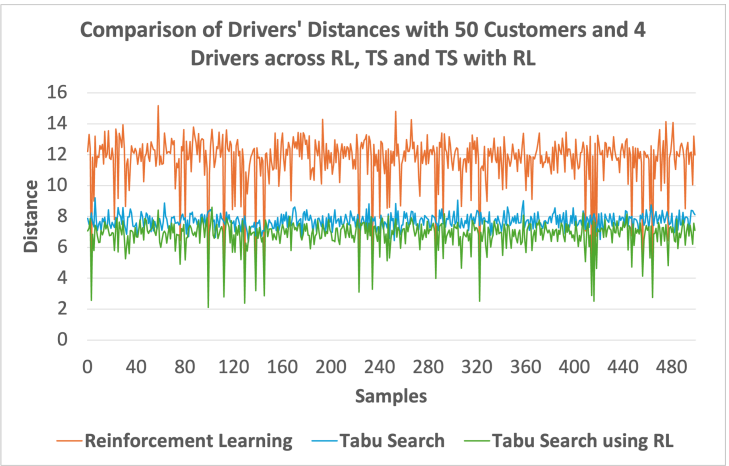


(b) 5 Drivers

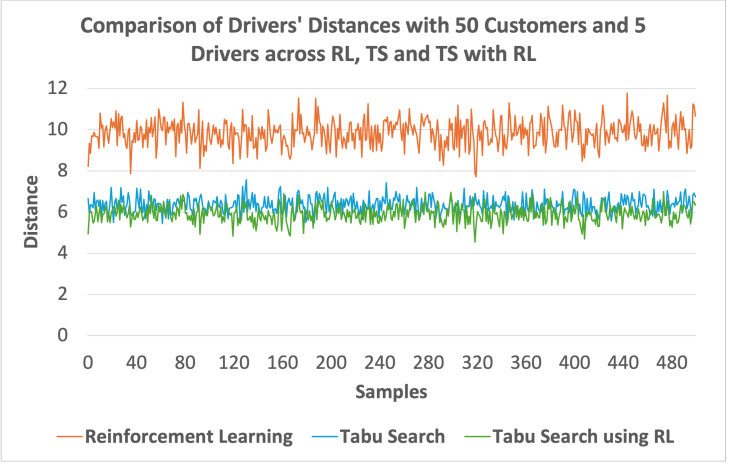


(c) 6 Drivers

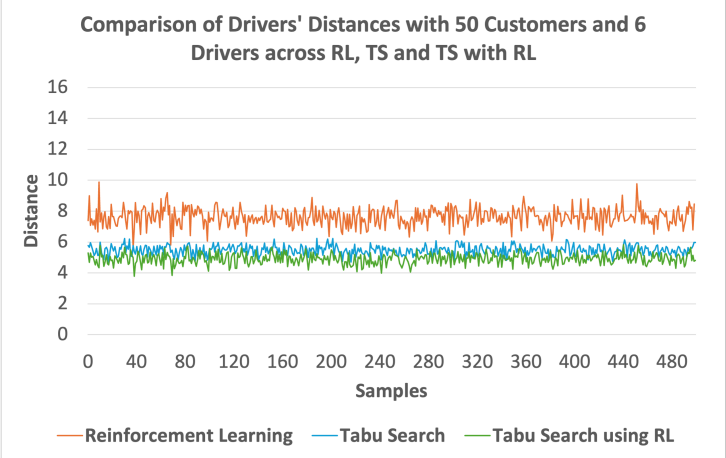
Figure 5.64 Line Chart comparing Drivers' Distances between RL, TS and TS with RL across samples with 20 customers



(a) 4 Drivers

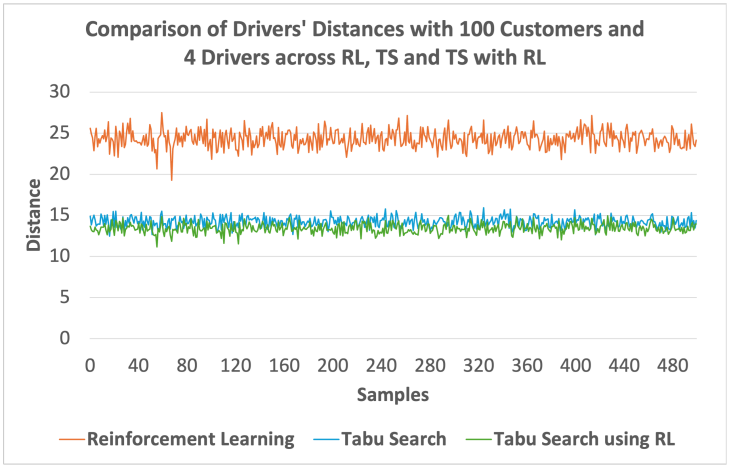


(b) 5 Drivers

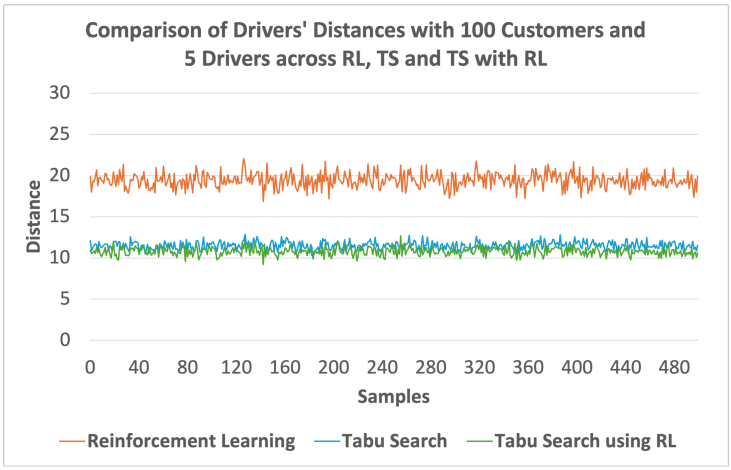


(c) 6 Drivers

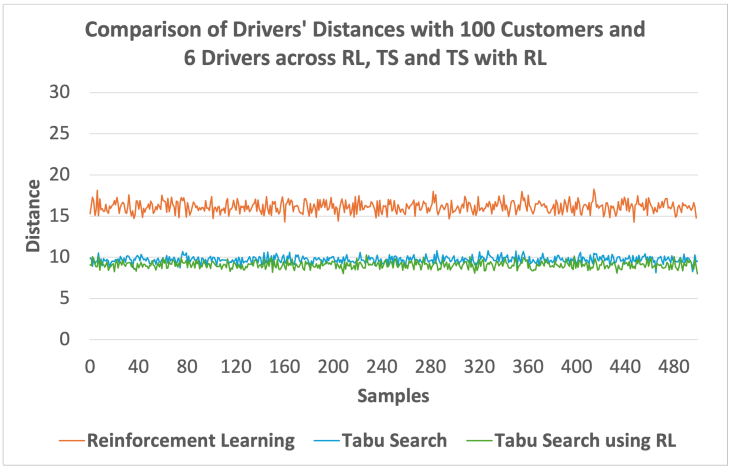
Figure 5.65 Line Chart comparing Drivers' Distances between RL, TS and TS with RL across samples with 50 customers



(a) 4 Drivers



(b) 5 Drivers



(c) 6 Drivers

Figure 5.66 Line Chart comparing Drivers' Distances between RL, TS and TS with RL across samples with 100 customers

Figures 5.67 to 5.69 show the average running time performance across all three models with groups of 20, 50 and 100 customers, respectively. The TS with RL is presented as a stacked bar to show the time taken to generate the initial solution using RL, followed by the time taken to use that initial solution to generate the final solution using TS.

Figure 5.67 shows that in instances with 20 customers, TS was the fastest and required the least computation time out of the three models, followed by RL and then TS with RL. The stacked bar belonging to TS with RL indicated that a large portion of its running time was due to RL taking a significant amount of time to generate the initial solution. However, in instances with 50 and 100 customers, as shown in Figure 5.68 and Figure 5.69, respectively, RL significantly outperformed the other models in terms of computational time, demonstrating its ability to handle larger problems more effectively by adapting to complex and dynamic environments. Both TS models performed similarly in terms of running time in these instances. They both exhibited a trend of increasing computation time with the addition of customers, indicating potential limitations in efficiently handling larger MVRRPP instances.

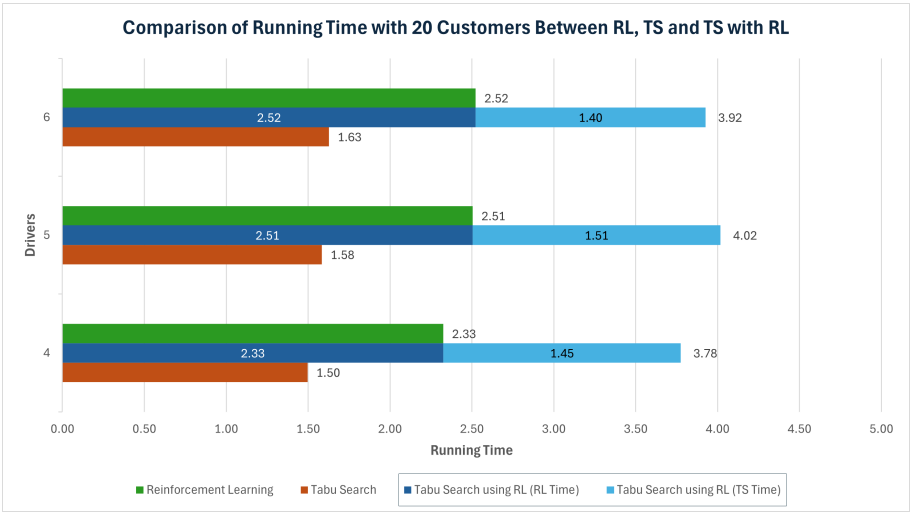


Figure 5.67 Comparing Average Running Time with 20 customers across each driver instance between RL, TS and TS with RL

The scatter plots in Figure 5.70 to 5.72 illustrate the performance of TS and TS-with-RL based on the time taken versus the number of iterations required to complete each sample in the dataset. The scatter plots show that the data points of each model formed a diagonal line, where the running time steadily increased as the number of iterations increased, indicating a consistent relationship between the two variables.

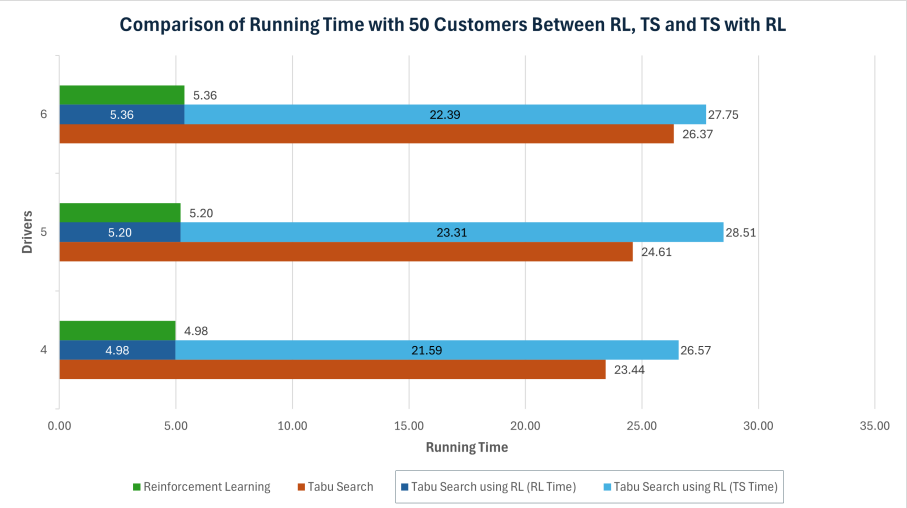


Figure 5.68 Comparing Average Running Time with 50 customers across each driver instance between RL, TS and TS with RL

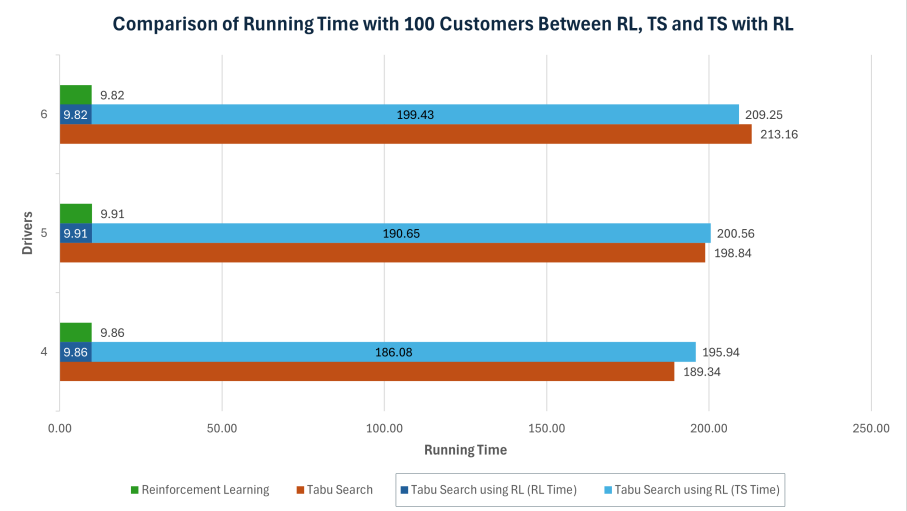
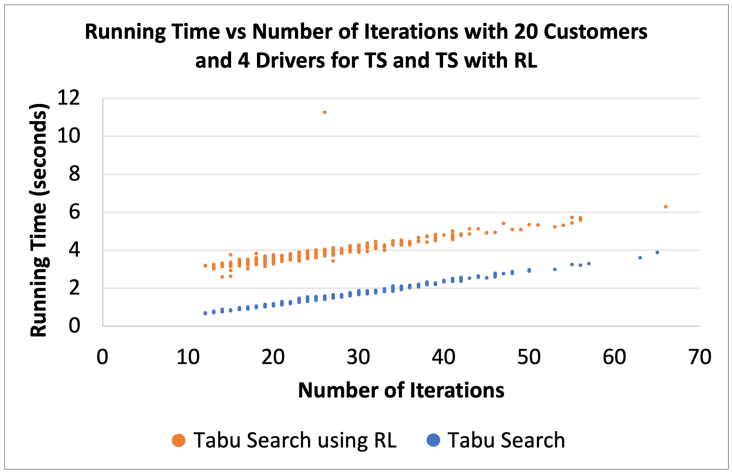
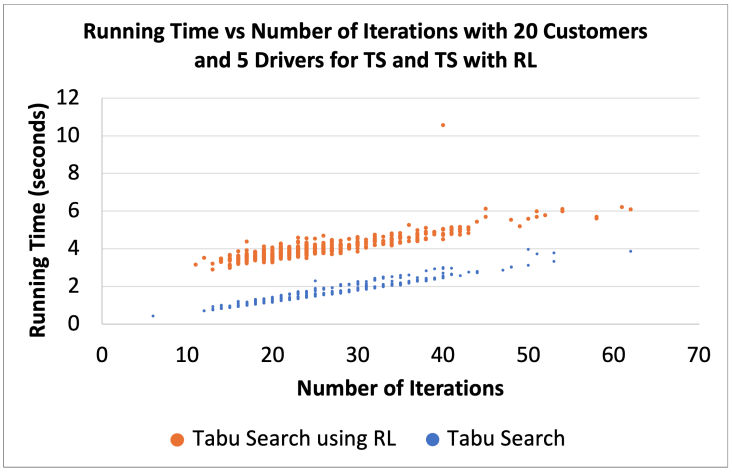


Figure 5.69 Comparing Average Running Time with 100 customers across each driver instance between RL, TS and TS with RL

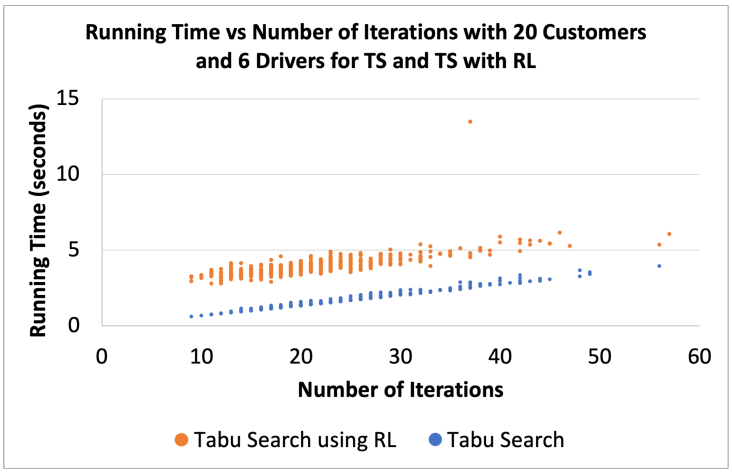
Lower diagonal lines belonging to those of TS, in instances of 20 customers and 50 customers, as shown in Figure 5.70 and Figure 5.71, respectively, indicate that the model completed the solution in less time using the same number of iterations, showing its efficiency over the TS with RL model. The scatter plots show that as the number of customers increases, the distance between the diagonal lines of each model decreases, suggesting a similar performance between models. This becomes apparent in instances with 100 customers, Figure 5.72, where data points of the two models overlap. Figure 5.72c shows that the data points belonging to TS with RL had lower overall values than TS, indicating that it was faster in finding the final solution.



(a) 4 Drivers

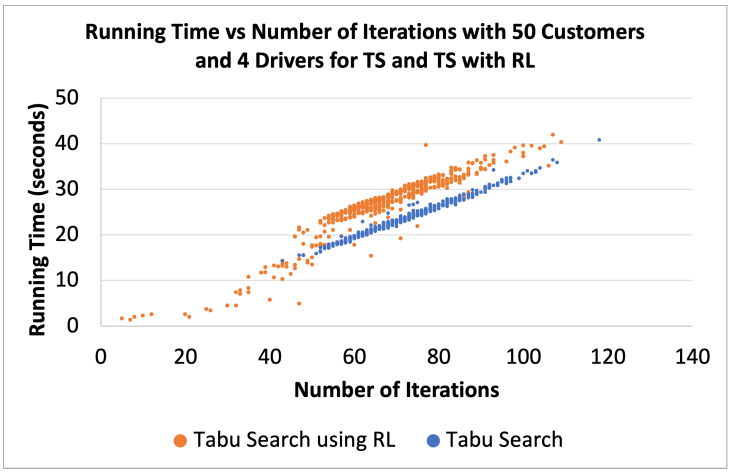


(b) 5 Drivers

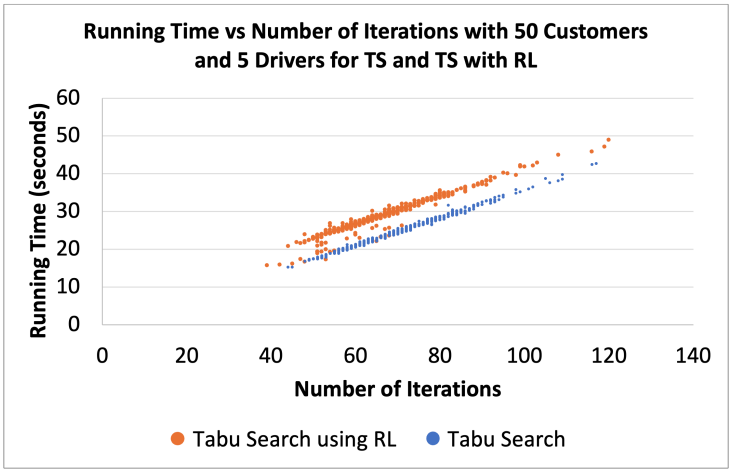


(c) 6 Drivers

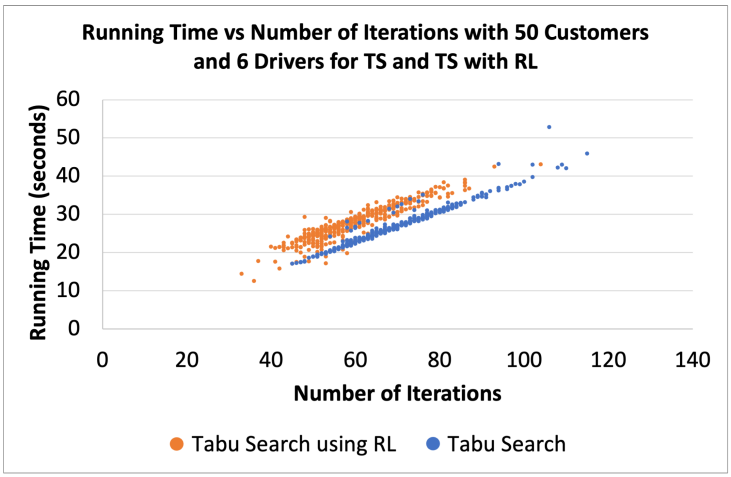
Figure 5.70 Scatter Plot comparing Running Time between RL, TS and TS with RL across samples with 20 customers



(a) 4 Drivers

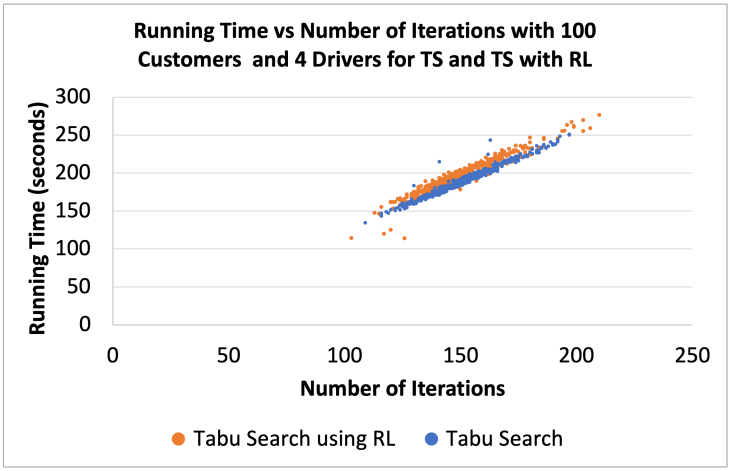


(b) 5 Drivers

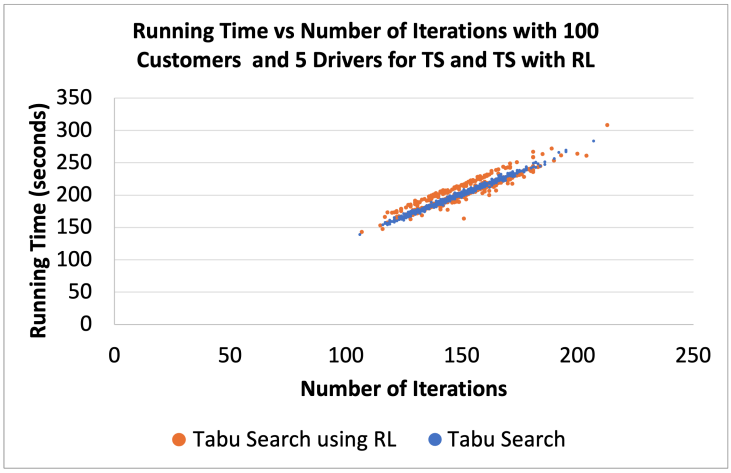


(c) 6 Drivers

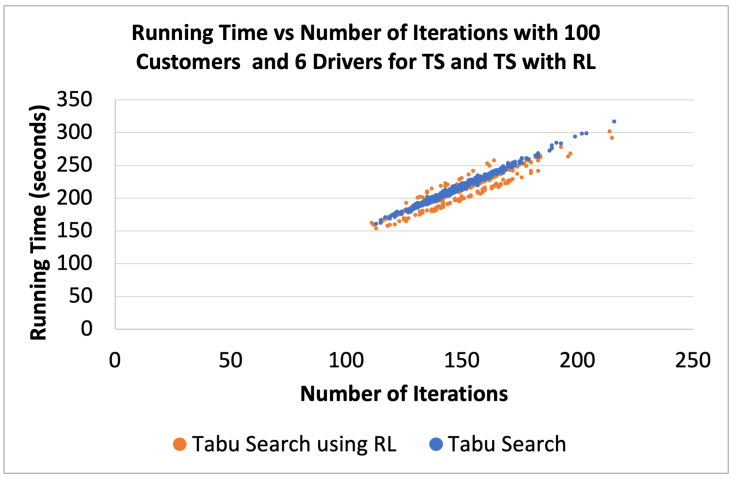
Figure 5.71 Scatter Plot comparing Running Time between RL, TS and TS with RL across samples with 50 customers



(a) 4 Drivers



(b) 5 Drivers



(c) 6 Drivers

Figure 5.72 Scatter Plot comparing Running Time between RL, TS and TS with RL across samples with 100 customers

TS is still very performant in terms of solution quality; as shown in the results, TS is hard to beat, which confirms why it is still being used in literature. However, TS's trade-off is that it can slow down significantly as the problem becomes larger with the addition of customers. Although RL's solution was less optimal, from a waiting time, travel impact time and distance perspective, it provided a quicker solution from an algorithmic complexity point. This shows that RL can provide a scalable solution at the cost of optimality.

The hybrid approach of using RL's output as the initial solution for TS showed minimal performance improvements in solution quality, with only occasional instances where it surpassed TS in terms of waiting time and travel impact time. However, as indicated by the distance metric, this hybrid approach was particularly effective in optimising the routing distances. The explanation for why TS with RL had a limited impact on performance and solution quality compared to TS could have been due to the quality of the initial solution generated by the RL model, as suggested by the RL model results. The initial solution produced by RL put the TS in a region of the solution space with a worse local optimum than the initial solution used in Experiment 1, where the customers were equally distributed. The fact that the TS with RL got stuck within this region indicates that the solution space was not convex.

In this area of research, RL has yet to match the performance of TS in terms of optimality. Although TS excelled in finding high-quality solutions, the trade-off of requiring significant computation time does not justify waiting to obtain a solution, making it less suitable for real-world scenarios with large-scale applications where efficiency is essential.

6 Conclusion

This research explored the challenges of developing a solution that optimises vehicle routing for ride-pooling, which is known to be a computationally challenging problem due to its NP-hard nature. While algorithms such as metaheuristics algorithms are already successful in solving such problems, they face challenges as the complexity and size of the problem increases.

6.1 Revisiting the Aim and Objectives

This research aimed to develop a reinforcement learning algorithm capable of solving the Multi-Vehicle Routing for Ride-Pooling Problem (MVRRPP). The problem determines the optimal set of routes by optimising passenger allocation to vehicles and vehicle routing. The RL algorithm aimed to generate these solutions faster than the traditional metaheuristic methods. To reach this aim, the following objectives were set:

6.1.1 Objective 1: Establishing a benchmark using Tabu Search

The first objective was achieved by implementing a metaheuristic benchmark algorithm using Tabu Search to solve the MVRRPP. For this algorithm, an initial solution containing the initial routes was created by equally distributing the customers to each driver's route in a round-robin fashion. The algorithm used the initial solution as a starting point to iteratively explore neighbourhood solutions within the solution space and find the best candidate. The best candidate cost was compared to the best solution using a cost function, where if the cost was lower, it was replaced as the new best solution. The purpose of the cost function was to minimise the total customers' waiting time and travel time and the total distance travelled by all vehicles. This process was repeated for several iterations where, in each iteration, the best candidate of the previous iteration was used to generate new neighbourhood solutions.

The results of the first objective indicated that each metric, consisting of the waiting time, travel impact time and driver distances increased as the number of customers was added to an instance. With the addition of customers, the driver had to make additional stops to pick up and drop off passengers, leading to a longer waiting time for customers to be picked up by the vehicle. The distance travelled by the vehicle also increases as the vehicle has to visit these extra stops. This also increased the travel impact time since additional stops may be made between a customer's pickup and dropoff location. In contrast, the results showed that these metrics decreased as the number of drivers was added to an instance, indicating that the algorithm successfully

distributed the customers along the drivers to reduce each metric. The results also showed that when using TS to solve instances with larger customers, the neighbourhood became more extensive and complex, requiring more iterations to converge to a local optimum and increasing computational time.

6.1.2 Objective 2: Implement a Reinforcement Learning algorithm for the MVRRPP

Objective 2 was implemented by using reinforcement learning to solve the MVRRPP. This was achieved by utilising the REINFORCE algorithm with a dynamic attention model adapted from literature by Peng et al. [75] to include the constraints defined in MVRRPP.

For this algorithm, the MVRRPP was modelled as RL problem, where the state, action and reward function were defined. The state consisted of the coordinates for the depots, pickup and dropoff locations, the available customers' pickup and dropoff locations, the current driver's origin location (for the current route) and the number of occupied seats for the current vehicle. The action space consisted of the set of possible locations the vehicle (agent) can select from at the next step. The reward function consisted of the cost function, which aimed at minimising the waiting time, travel time, and distance travelled in a solution. A penalty for the number of customers left unpicked in an instance was also added to the reward function.

The RL model used a dynamic attention model with a dynamic encoder-decoder architecture. An encoder was used to encode information of the MVRRPP representation, such as the coordinates of locations, to an embedding. A multi-head attention mechanism was implemented in the encoder to focus the attention of the embeddings on unvisited nodes. The decoder was then implemented to use the encoder's embeddings and the current environment's state as input for the decoder's multi-head attention mechanism, which was responsible for understanding the current state and the relationship between nodes required for determining the vehicles' optimal routes. A mask was created for the decoder's attention mechanism to ensure the agent selected a valid node as its next action. This was done by excluding (masking out) invalid actions that do not adhere to the constraints and rules of the MVRRPP. The decoder was then responsible for generating a probability distribution over the possible actions for the agent to determine which location to select next.

The results from this experiment showed that, similarly to TS, the waiting time, travel impact time and distance increased with the addition of customers. Adding more drivers did not necessarily improve waiting time and travel impact time, unlike TS, suggesting that the RL model might have found a difficulty in balancing the trade-off between waiting time and travel impact time. When evaluating the distance metrics, RL

showed that adding more drivers reduced overall distance, indicating that the RL model successfully optimised the routes based on the proximity of locations and effectively distributed customers across routes to minimise the overall distance travelled. When comparing the performance of RL with that of TS, TS performed better with lower waiting time, travel impact time and distance across all instances. The results showed that while TS was faster when solving smaller problems, such as instances with 20 customers, it struggled with larger problems as the number of customers increased with the TS computation time increasing exponentially compared to that of RL.

6.1.3 Objective 3: Initialising Tabu Search with a solution obtained using Reinforcement Learning

Objective 3 was achieved by using the output solution generated by the RL model as the initial solution for the TS model. The objective aimed to determine whether combining RL with TS made the optimisation process faster and improved the MVRRPP solution.

The results showed similar waiting time and travel impact time performance between TS-with-RL and TS, with only a few instances where TS-with-RL surpassed TS. In contrast, RL exhibited the highest waiting time and travel impact time across the two metrics, indicating challenges in optimising vehicle routing solutions effectively. When evaluating the distance metric, TS with RL outperformed the other models with the least distance travelled across all driver instances with different numbers of customers. TS had a slightly higher distance across instances but was still considerably lower than RL's. The results showed that in terms of running time to generate the optimal solution, TS performed the fastest in smaller problems, with TS-with-RL taking the longest, while RL was the fastest to generate solutions in larger problems. Both TS models showed a substantial increase in computation time as the number of customers increased, suggesting limitations in the efficiency for larger instances.

6.2 Summary of Findings

The RL and TS experiments conducted during this research showed that when solving the MVRRPP, TS effectively found higher-quality solutions in terms of waiting time, travel impact time and total distance travelled compared to RL. This confirms why in literature [1] and [60], TS is still a popular algorithm.

The results demonstrated how the TS implementation successfully explored the solution space by iteratively exploring neighbouring solutions and using a tabu list to avoid revisiting previously visited solutions. This allowed it to search for unexplored

regions of the solution space and find the best possible solutions. The trade-off of TS is its computational complexity, resulting in longer computation times when solving large problem instances consisting of many customers. This becomes problematic in real-world ride-pooling systems where the time to generate a solution in real time is crucial. Although the results indicate that using RL involves a trade-off between solution quality and computation time, it was quicker at finding a solution from an algorithmic complexity point, making it a suitable alternative for finding a scalable solution.

Using TS with RL showed minimal performance gains regarding solution quality, with a few instances where it surpassed TS in terms of waiting time, travel impact time and running time. However, when evaluating distance, TS with RL showed the most effectiveness in performance as it was successful in reducing the overall distance, indicating that the initial solution produced using RL could have focused on optimising distance. The fact that the solution generated by TS-with-RL was not reaching the same solution quality as that of TS indicates that the initial solution generated by RL was in a region of the solution space that had a worse local optimum than the TS implementation containing an initial solution with equally distributed customers between drivers. The TS-with-RL solution being stuck in a region with a worse local optimum suggests that the solution space was non-convex.

Therefore, the choice between TS and RL for MVRRPP depends on the specific requirements of the use-case scenario, balancing the need for high-quality solutions against the available computational time.

6.3 Future Work

While most existent research focuses on using metaheuristic algorithms to solve the VRP, this research demonstrated the possibility of implementing RL to solve a more complex variant of the VRP, known as the MVRRPP, paving the way for future work in this field of research.

Further research may be carried out using alternative RL algorithms to solve the MVRRPP, such as DQN, which utilises Q-learning to estimate the cost function value of taking specific actions in particular states, allowing it to generalise similar states and handle large MVRRPP instances. Additional constraints for the MVRRPP may also be explored, such as passenger constraints, where passengers may specify the departure and arrival times, requiring the algorithm to reconsider which vehicles to pick up first while minimising the other customers' waiting time and travel time.

Another possible research area is adapting the MVRRPP to use Electric Vehicles (EVs) rather than vehicles with an internal combustion engine. This problem would

form part of the Electric Vehicle Routing Problem (EVRP), which includes additional operations such as recharging and minimising the total energy consumption due to the battery limitations that limit EVs' driving range. When EVs are used for ride-pooling, the algorithm must determine if there is enough battery charge for the vehicle to reach the pickup and dropoff locations and return to the origin location. If the vehicle doesn't have enough battery for any of those locations, then a different vehicle is allocated to that customer. The EV must always save additional battery for returning to the origin location, where they can recharge their vehicles. Further research can be carried out to explore the effect of the algorithm when optimising routes for EVs with different battery capacities.

6.4 Final Remarks

As ride-pooling services surged in popularity over the last couple of years, there is a need for algorithms to solve the allocation of vehicles for customers' requests and routing optimisation in real time. In this research, we explored the possibility of using a Reinforcement Learning algorithm to solve the Multi-Vehicle Routing for Ride-Pooling Problem, a combinatorial problem typically solved using metaheuristics. We successfully implemented an algorithm for this problem using the REINFORCE algorithm with a dynamic attention model consisting of a dynamic encoder-decoder architecture that can generate solutions faster than traditional metaheuristic algorithms, allowing it to be implemented in real-world systems. While the solution quality of this approach is still inferior to that of the metaheuristics algorithms such as Tabu Search, generating a solution with Reinforcement Learning is significantly faster as the input size grows, offering a real-time solution for this problem.

References

[1] G. Li and J. Li, "An improved tabu search algorithm for the stochastic vehicle routing problem with soft time windows," *IEEE Access*, vol. 8, pp. 158 115–158 124, 2020. DOI: 10.1109/ACCESS.2020.3020093.

[2] B. Rabbouch, F. Saâdaoui, and R. Mraihi, "Efficient implementation of the genetic algorithm to solve rich vehicle routing problems," *Operational Research*, vol. 21, pp. 1763–1791, 2021. DOI: 10.1007/s12351019005210.

[3] J. M. Vera and A. G. Abad, "Deep reinforcement learning for routing a heterogeneous fleet of vehicles," in *2019 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, 2019, pp. 1–6. DOI: 10.1109/LA-CCI47412.2019.9037042.

[4] K. Zhang, F. He, Z. Zhang, X. Lin, and M. Li, "Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach," *Transportation Research Part C: Emerging Technologies*, vol. 121, p. 102 861, 2020, ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2020.102861.

[5] T. Mustakhov, Y. Akhmetbek, and A. Bogyrbayeva, "Deep reinforcement learning for stochastic dynamic vehicle routing problem," in *2023 17th International Conference on Electronics Computer and Computation (ICECCO)*, 2023, pp. 1–5. DOI: 10.1109/ICECCO58239.2023.10147154.

[6] G. Laporte, "What you should know about the vehicle routing problem," *Naval Research Logistics (NRL)*, vol. 54, no. 8, pp. 811–819, 2007. DOI: https://doi.org/10.1002/nav.20261.

[7] R. Basso, B. Kulcsár, I. Sanchez-Diaz, and X. Qu, "Dynamic stochastic electric vehicle routing with safe reinforcement learning," *Transportation Research Part E: Logistics and Transportation Review*, vol. 157, p. 102 496, 2022, ISSN: 1366-5545. DOI: https://doi.org/10.1016/j.tre.2021.102496.

[8] J. Zhao, M. Mao, X. Zhao, and J. Zou, "A hybrid of deep reinforcement learning and local search for the vehicle routing problems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 11, pp. 7208–7218, 2021. DOI: 10.1109/TITS.2020.3003163.

[9] P. Toth and D. Vigo, *The Vehicle Routing Problem*, P. Toth and D. Vigo, Eds. Society for Industrial and Applied Mathematics, 2002. DOI: 10.1137/1.9780898718515.

[10] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959, ISSN: 00251909, 15265501.

[11] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse, "The vehicle routing problem: State of the art classification and review," *Computers & Industrial Engineering*, vol. 99, pp. 300–313, 2016, ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2015.12.007.

[12] S.-Y. Tan and W.-C. Yeh, "The vehicle routing problem: State-of-the-art classification and review," *Applied Sciences*, vol. 11, no. 21, 2021, ISSN: 2076-3417. DOI: 10.3390/app112110295.

[13] G. D. Konstantakopoulos, S. P. Gayialis, and E. P. Kechagias, "Vehicle routing problem and related algorithms for logistics distribution: A literature review and classification," eng, *Operational research*, vol. 22, no. 3, pp. 2033–2062, 2022, ISSN: 1109-2858.

[14] R. Necula, M. Breaban, and M. Raschip, "Tackling dynamic vehicle routing problem with time windows by means of ant colony system," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, 2017, pp. 2480–2487. DOI: 10.1109/CEC.2017.7969606.

[15] H. Park, D. Son, B. Koo, and B. Jeong, "Waiting strategy for the vehicle routing problem with simultaneous pickup and delivery using genetic algorithm," *Expert Systems with Applications*, vol. 165, p. 113 959, 2021, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.113959.

[16] W. Ho, G. T. Ho, P. Ji, and H. C. Lau, "A hybrid genetic algorithm for the multi-depot vehicle routing problem," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 4, pp. 548–557, 2008, ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2007.06.001.

[17] H. QIN, X. SU, T. REN, and Z. LUO, "A review on the electric vehicle routing problems: Variants and algorithms," eng, *Frontiers of Engineering Management*, vol. 8, no. 3, pp. 370–389, 2021, ISSN: 2095-7513.

[18] S. Karakatič, "Optimizing nonlinear charging times of electric vehicle routing with genetic algorithm," *Expert Systems with Applications*, vol. 164, p. 114 039, 2021, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.114039.

[19] S. Zhang, Y. Gajpal, S. Appadoo, and M. Abdulkader, "Electric vehicle routing problem with recharging stations for minimizing energy consumption," *International Journal of Production Economics*, vol. 203, pp. 404–413, 2018, ISSN: 0925-5273. DOI: https://doi.org/10.1016/j.ijpe.2018.07.016.

[20] M. Mavrovouniotis, C. Li, G. Ellinas, and M. Polycarpou, "Parallel ant colony optimization for the electric vehicle routing problem," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2019, pp. 1660–1667. DOI: 10.1109/SSCI44817.2019.9003153.

[21] T. Dokeroglu, E. Sevinc, T. Kucukyilmaz, and A. Cosar, "A survey on new generation metaheuristic algorithms," *Computers & Industrial Engineering*, vol. 137, p. 106 040, 2019, ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2019.106040.

[22] M. Abdel-Basset, L. Abdel-Fatah, and A. K. Sangaiah, "Metaheuristic algorithms: A comprehensive review," in *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications*, ser. Intelligent Data-Centric Systems, A. K. Sangaiah, M. Sheng, and Z. Zhang, Eds., Academic Press, 2018, pp. 185–231, ISBN: 978-0-12-813314-9. DOI: https://doi.org/10.1016/B978-0-12-813314-9.00010-4.

[23] A. H. Gandomi, X.-S. Yang, S. Talatahari, and A. H. Alavi, *Metaheuristic Applications in Structures and Infrastructures*, 1st ed. Elsevier, Jan. 2013, ISBN: 978-0-12-398364-0.

[24] P. Agrawal, H. F. Abutarboush, T. Ganesh, and A. W. Mohamed, "Metaheuristic algorithms on feature selection: A survey of one decade of research (2009-2019)," *IEEE Access*, vol. 9, pp. 26 766–26 791, 2021. DOI: 10.1109/ACCESS.2021.3056407.

[25] A. Amuthan and K. Deepa Thilak, "Survey on tabu search meta-heuristic optimization," in *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, 2016, pp. 1539–1543. DOI: 10.1109/SCOPES.2016.7955697.

[26] F. Glover and M. Laguna, "Tabu search," in *Handbook of Combinatorial Optimization: Volume1–3*, D.-Z. Du and P. M. Pardalos, Eds. Boston, MA: Springer US, 1998, pp. 2093–2229, ISBN: 978-1-4613-0303-9. DOI: 10.1007/978-1-4613-0303-9_33.

[27] E. Aarts, J. Korst, and W. Michiels, "Simulated annealing," in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. Burke and G. Kendall, Eds. Boston, MA: Springer US, 2014, pp. 265–285, ISBN: 978-1-4614-6940-7. DOI: 10.1007/978-1-4614-6940-7_10.

[28] D. Henderson, S. H. Jacobson, and A. W. Johnson, "The theory and practice of simulated annealing," in *Handbook of Metaheuristics*, F. Glover and G. A. Kochenberger, Eds. Boston, MA: Springer US, 2003, pp. 287–319, ISBN: 978-0-306-48056-0. DOI: 10.1007/0-306-48056-5_10.

[29] Y. Kaya, M. Uyar, and R. Tekjn, *A novel crossover operator for genetic algorithms: Ring crossover*, 2011. arXiv: 1105.0355 [cs.NE].

[30] M. Bramer, *Principles of Data Mining*. Springer London, 2020. DOI: 10.1007/978-1-4471-7493-6.

[31] J. Patterson and A. Gibson, *Deep Learning : A Practitioner's Approach*. O'Reilly Media, Inc., 2017.

[32] J. D. Kelleher, *Deep Learning*. The MIT Press, 2019.

[33] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. DOI: 10.1038/nature14539.

[34] L. Zhang, S. Wang, and B. Liu, *Deep learning for sentiment analysis : A survey*, 2018. arXiv: 1801.07883 [cs.CL].

[35] A. Vaswani *et al.*, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL].

[36] F. Xia *et al.*, "Graph learning: A survey," *IEEE Transactions on Artificial Intelligence*, vol. 2, no. 2, pp. 109–127, 2021. DOI: 10.1109/TAI.2021.3076021.

[37] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021. DOI: 10.1109/TNNLS.2020.2978386.

[38] N. A. Asif *et al.*, "Graph neural network: A comprehensive review on non-euclidean space," *IEEE Access*, vol. 9, pp. 60 588–60 606, 2021. DOI: 10.1109/ACCESS.2021.3071274.

[39] L. Wu, P. Cui, J. Pei, and L. Zhao, *Graph Neural Networks: Foundations, Frontiers, and Applications*, 1st ed. Springer Singapore, Jan. 2022, ISBN: 978-981-16-6053-5. DOI: 10.1007/978-981-16-6054-2.

[40] J. Zhou *et al.*, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020, ISSN: 2666-6510. DOI: https://doi.org/10.1016/j.aiopen.2021.01.001.

[41] G. Wang, R. Ying, J. Huang, and J. Leskovec, *Improving graph attention networks with large margin-based constraints*, 2019. arXiv: 1910.11945 [cs.LG].

[42] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, *Graph attention networks*, 2018. arXiv: 1710.10903 [stat.ML].

[43] M. Naeem, S. T. H. Rizvi, and A. Coronato, "A gentle introduction to reinforcement learning and its application in different fields," *IEEE Access*, vol. 8, pp. 209 320–209 344, 2020. DOI: 10.1109/ACCESS.2020.3038605.

[44] A. Plaat, *Deep Reinforcement Learning*. Singapore: Springer, 2022, ISBN: 9789811906374.

[45] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017. DOI: 10.1109/MSP.2017.2743240.

[46] R. S. Sutton and A. Barto, *Reinforcement learning: An Introduction*, Second edition. Cambridge, Massachusetts ; London, England: The MIT Press, 2018, ISBN: 9780262039246.

[47] P. Ladosz, L. Weng, M. Kim, and H. Oh, "Exploration in deep reinforcement learning: A survey," *Information Fusion*, vol. 85, pp. 1–22, 2022, ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2022.03.003.

[48] A. Alharin, T.-N. Doan, and M. Sartipi, "Reinforcement learning interpretation methods: A survey," *IEEE Access*, vol. 8, pp. 171 058–171 077, 2020. DOI: 10.1109/ACCESS.2020.3023394.

[49] Z. Zhang, D. Zhang, and R. C. Qiu, "Deep reinforcement learning for power system applications: An overview," *CSEE Journal of Power and Energy Systems*, vol. 6, no. 1, pp. 213–225, 2020. DOI: 10.17775/CSEEJPES.2019.00920.

[50] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[51] H. Dong, Z. Ding, and S. Zhang, *Deep Reinforcement Learning Fundamentals, Research and Applications*. Singapore: Springer Singapore, 2020, ISBN: 9789811540950.

[52] R. Nian, J. Liu, and B. Huang, "A review on reinforcement learning: Introduction and applications in industrial process control," *Computers & Chemical Engineering*, vol. 139, p. 106 886, 2020, ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2020.106886.

[53] J. Zhang, J. Kim, B. O'Donoghue, and S. Boyd, *Sample efficient reinforcement learning with reinforce*, 2020. arXiv: 2010.11364 [cs.LG].

[54] F. AlMahamid and K. Grolinger, "Reinforcement learning algorithms: An overview and classification," in *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, IEEE, Sep. 2021. DOI: 10.1109/ccece53047.2021.9569056.

[55] E. Noorani and J. S. Baras, "Risk-sensitive reinforce: A monte carlo policy gradient algorithm for exponential performance criteria," in *2021 60th IEEE Conference on Decision and Control (CDC)*, 2021, pp. 1522–1527. DOI: 10.1109/CDC45484.2021.9683645.

[56] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, pp. 229–256, 1992.

[57] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. DOI: https://doi.org/10.1038/nature14236.

[58] H. van Hasselt, A. Guez, and D. Silver, *Deep reinforcement learning with double q-learning*, 2015. arXiv: 1509.06461 [cs.LG].

[59] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG].

[60] M. Gmira, M. Gendreau, A. Lodi, and J.-Y. Potvin, "Tabu search for the time-dependent vehicle routing problem with time windows on a road network," *European Journal of Operational Research*, vol. 288, no. 1, pp. 129–140, 2021, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2020.05.041.

[61] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin, "A tabu search heuristic for the vehicle routing problem with soft time windows," *Transportation Science*, vol. 31, pp. 170–186, May 1997. DOI: 10.1287/trsc.31.2.170.

[62] M. Polacek, R. F. Hartl, K. Doerner, and M. Reimann, "A variable neighborhood search for the multi depot vehicle routing problem with time windows," *Journal of Heuristics*, vol. 10, no. 6, pp. 613–627, 2004. DOI: https://doi.org/10.1007/s1073200554325.

[63] J.-F. Cordeau, G. Laporte, and A. Mercier, "A unified tabu search heuristic for vehicle routing problems with time windows," *The Journal of the Operational Research Society*, vol. 52, no. 8, pp. 928–936, 2001, ISSN: 01605682, 14769360.

[64] V. F. Yu, H. Susanto, P. Jodiawan, T.-W. Ho, S.-W. Lin, and Y.-T. Huang, "A simulated annealing algorithm for the vehicle routing problem with parcel lockers," *IEEE Access*, vol. 10, pp. 20 764–20 782, 2022. DOI: 10.1109/ACCESS.2022.3152062.

[65] Y. Lin, W. Li, F. Qiu, and H. Xu, "Research on optimization of vehicle routing problem for ride-sharing taxi," *Procedia - Social and Behavioral Sciences*, vol. 43, pp. 494–502, 2012, 8th International Conference on Traffic and Transportation Studies (ICTTS 2012), ISSN: 1877-0428. DOI: https://doi.org/10.1016/j.sbspro.2012.04.122.

[66] W. M. Herbawi and M. Weber, "A genetic and insertion heuristic algorithm for solving the dynamic ridematching problem with time windows," in *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '12, Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 2012, pp. 385–392, ISBN: 9781450311779. DOI: 10.1145/2330163.2330219.

[67] A. O. Al-Abbasi, A. Ghosh, and V. Aggarwal, "Deeppool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 12, pp. 4714–4727, 2019. DOI: 10.1109/TITS.2019.2931830.

[68] G. Guo and Y. Xu, "A deep reinforcement learning approach to ride-sharing vehicle dispatching in autonomous mobility-on-demand systems," *IEEE Intelligent Transportation Systems Magazine*, vol. 14, no. 1, pp. 128–140, 2022. DOI: 10.1109/MITS.2019.2962159.

[69] J. Alonso-Mora, A. Wallar, and D. Rus, "Predictive routing for autonomous mobility-on-demand systems with ride-sharing," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 3583–3590. DOI: 10.1109/IROS.2017.8206203.

[70] A. Wren and A. Holliday, "Computer scheduling of vehicles from one or more depots to a number of delivery points," *Journal of the Operational Research Society*, vol. 23, no. 3, pp. 333–344, 1972. DOI: 10.1057/jors.1972.53.

[71] G. Clarke and J. W. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations research*, vol. 12, no. 4, pp. 568–581, 1964.

[72] W. Kool, H. van Hoof, and M. Welling, *Attention, learn to solve routing problems!* 2019. arXiv: 1803.08475 [stat.ML].

[73] K. Zhang, X. Lin, and M. Li, "Graph attention reinforcement learning with flexible matching policies for multi-depot vehicle routing problems," *Physica A: Statistical Mechanics and its Applications*, vol. 611, p. 128 451, 2023, ISSN: 0378-4371. DOI: https://doi.org/10.1016/j.physa.2023.128451.

[74] A. Gupta, S. Ghosh, and A. 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)*, ser. CODS-COMAD 2022, Bangalore, India: Association for Computing Machinery, 2022, pp. 236–240, ISBN: 9781450385824. DOI: 10.1145/3493700.3493723.

[75] B. Peng, J. Wang, and Z. Zhang, *A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems*, 2020. arXiv: 2002.03282 [cs.LG].