University of Bergen
Department of Informatics

A Deep Reinforcement Learning - based Hyperheuristic for the Flexible Traveling Repairman Problem with Drones

Author: Vegard Øverland Birkenes
Supervisor: Ahmad Hemmati

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

July, 2023

Abstract

The Flexible Traveling Repairman Problem with Drones (FTRPD) is a combinatorial optimization problem concerned with optimizing delivery routes in a multi-modal system combining a truck and Unmanned Aerial Vehicle (UAV) operations. In this system, a truck and UAV operations are synchronized, i.e., one or more UAV travel on a truck, which serves as a mobile depot. Deliveries can be made by both UAVs and the truck. While the truck follows a multi-stop route, each UAV delivers a single shipment per dispatch. This is a practical problem with direct applications in the logistics industry, particularly for last-mile delivery. In this thesis, we implement and apply two fundamental techniques - metaheuristics and reinforcement learning - to the FTRPD. The Truck and Drone Routing Algorithm (TDRA) is a metaheuristic algorithm inspired by Adaptive Large Neighborhood Search (ALNS). Deep Reinforcement Learning Hyperheuristic (DRLH) is a general framework for heuristic selection based on Deep Reinforcement Learning (DRL), replacing the adaptive layer of ALNS. The goal is to compare how the heuristic selection of the approaches affects performance. The approaches are evaluated across two baseline sets with problem sizes 10, 20, and 50. Our results show that DRLH becomes increasingly effective over TDRA as the problem size increases. Furthermore, DRLH performs more consistently and, in terms of average objective, converges in fewer iterations than TDRA.

Acknowledgements

First and foremost, I would like to thank my supervisor, Ahmad Hemmati, for his continued support and great guidance. Thanks to my parents and family for their enduring encouragement. I appreciate my friends and my fellow machine learning students for the inspiring conversations and camaraderie, making this journey an enriching experience. Special thanks to my girlfriend, Henriette, for her constant love and patience. This achievement is the result of the collective efforts and faith of all these wonderful people in my life. Thank you.

Vegard Øverland Birkenes
Saturday 1st July, 2023

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Combinatorial optimization is an intricate field of study that requires robust and adaptable algorithms to solve its diverse range of problems. This thesis focuses on applying two fundamental techniques – metaheuristics and reinforcement learning – in the context of the Flexible Traveling Repairman Problem with Drones (FTRPD).

Metaheuristics are high-level strategies that have proven useful in a variety of optimization problems across different fields, including logistics (Moshref-Javadi et al., 2020), telecommunications (Alvarez et al., 2018), and computational biology (Shukla et al., 2020). Meanwhile, reinforcement learning, a subset of machine learning, offers an approach to learning decision-making processes based on reward feedback and has shown potential in complex, sequential decision-making scenarios. In this thesis, we focus on two specific methods, Truck and Drone Routing Algorithm (TDRA) and Deep Reinforcement Learning Hyperheuristic (DRLH), both of which derive their inspiration from Adaptive Large Neighborhood Search (ALNS). TDRA is an adaptation of ALNS explicitly designed for the FTRPD, while DRLH is a deep reinforcement learning agent devised to replace the adaptive layer of ALNS.

We focus on the Flexible Traveling Repairman Problem with Drones (FTRPD), an instance of a combinatorial optimization problem. This problem is centered on finding an efficient route for a vehicle and its accompanying drones to deliver packages. This is a practical challenge with direct applications in the logistics industry, particularly in last-mile deliveries.

Our goal is to compare the effectiveness of TDRA and DRLH when applied to the FTRPD. In addition, we aim to gain insights into each method's advantages and potential limitations. This is achieved through careful experimentation and assessment of the method's performance across various established metrics.

1.2 Thesis Outline

The outline of the rest of the thesis is as follows.

Chapter 2 – Background gives the theoretical background related to combinatorial optimization, metaheuristics, and reinforcement learning required for this thesis. It also covers related work on last-mile delivery with autonomous assistants, hyperheuristics, and solving combinatorial optimization problems using deep reinforcement learning.

Chapter 3 – Flexible Traveling Repairman Problem with Drones describes the dynamics of the FTRPD and the Truck and Drone Routing Algorithm (TDRA).

Chapter 4 – Deep Reinforcement Learning Hyperheuristic describes the dynamics of the DRLH framework.

Chapter 5 – Experimental Setup contains how the experiments were conducted. This includes hardware to run the experiments, information about baseline methods, parameters, and training set generation.

Chapter 6 – Results describe the findings of the experiments and discuss their relevance and significance.

Chapter 7 – Conclusion and Future Work summarizes and concludes the thesis and looks at potential future work related to the thesis.

Chapter 2

Background and Related Work

2.1 Combinatorial Optimization Problems

Combinatorial optimization problems are a class of mathematical problems that involve finding the best arrangement or selection of discrete objects from a finite set, subject to certain constraints Schrijver (2003). The goal is to optimize some objective function that measures the quality of the chosen arrangement or selection. Combinatorial optimization problems arise in several fields, including computer science, engineering, economics, and operations research. Some classic examples of combinatorial optimization problems include the Traveling Salesman Problem (TSP), where one seeks the shortest possible route that visits a set of cities exactly once, and the knapsack problem, where one seeks the most valuable set of items that can fit in a limited-capacity container. Combinatorial optimization problems can be extremely challenging to solve, and various techniques such as heuristics, exact algorithms, and approximation algorithms have been developed to tackle them.

One of the reasons they can be difficult to solve is the many possible combinations that must be explored to find the optimal solution. This is especially true for problems of larger sizes, where it might be infeasible to enumerate all possible solutions and select the best one. Moreover, many combinatorial optimization problems are NP-hard, meaning no known algorithm can solve them in polynomial time. This implies that the best-known algorithms for solving these problems require time that grows exponentially with the size of the problem instance in the worst case. While many heuristic and approximation algorithms have been developed to find high-quality solutions to many combinatorial optimization problems in a reasonable time, no general algorithm can solve all such problems efficiently. As a result, solving combinatorial optimization problems often requires a combination of algorithmic ingenuity, domain-specific knowledge, and careful modeling of the problem.

2.2 Solution Methods

Various solution methods for combinatorial optimization problems can be employed based on the specific problem instance and the desired solution quality. Exact methods, heuristic methods, and metaheuristics are three broad categories of solution methods used to tackle combinatorial optimization problems. Exact methods provide optimality guarantees but may be computationally expensive, while heuristic methods trade optimality for speed and can be helpful in large-scale problems. Metaheuristics offer a middle ground between exact and heuristic methods, using high-level strategies to guide the search for good solutions. Each of these solution methods has its advantages and disadvantages, and the choice of method will depend on the specific problem instance and the requirements of the problem at hand.

2.2.1 Exact Methods

Exact methods are algorithms guaranteed to find the optimal solution to a problem. Unfortunately, they are often computationally expensive and impractical for large-scale problems. This is because a sizable portion of the solution space must be explored. In addition, exact methods are often hard to use on real-life problems due to many constraints and large instance sizes. Some examples of exact methods include branch-and-bound, Dynamic Programming, and Integer Programming.

2.2.2 Heuristic Methods

Heuristic methods are a family of optimization algorithms that aim to find good solutions to combinatorial optimization problems in a reasonable amount of time. Unlike exact methods, which guarantee to find the optimal solution but can be computationally expensive, heuristic methods provide an approximate solution that may not be optimal but is often good enough for practical purposes. Heuristic methods can be broadly classified into constructive and perturbative heuristics. Constructive heuristics build a solution step-by-step, starting with an empty solution and adding elements. Meanwhile, perturbative heuristics begin with an initial solution and make minor random modifications to it. Heuristic methods are often used when the instance size or complexity is too large for exact methods to handle or when the exact solution is unnecessary.

Constructive Heuristics

Constructive heuristics build a solution step-by-step, starting from an empty solution and adding elements until a complete solution is obtained. Although these algorithms are often simple and easy to implement, they can provide good solutions for small and medium-sized instances. Because of this, perturbative heuristics often use a constructive heuristic to build an initial, feasible solution. An example of a constructive heuristic is greedy constructive. This method selects the best option at each step according to some criterion, such as minimizing the cost or maximizing the profit.

Perturbative Heuristics

Perturbative heuristics start with an initial solution and make minor random modifications to it to explore new regions of the search space. These algorithms are often more complex and computationally intensive than constructive heuristics, but they can provide better solutions for larger and more complex problems.

An essential aspect of perturbative heuristics is *neighborhoods*. The neighborhood of a given solution is a set of all the solutions that can be obtained by applying a small modification to the original solution. To avoid getting stuck in a local optimum, the size of the neighborhood should be large enough to explore a wide range of solutions but small enough to avoid excessive computation. A large neighborhood can be achieved by having a good mix of *diversification* and *intensification*. Diversification involves exploring various solutions to avoid getting stuck in a local optimum. This might worsen the immediate objective in hopes of finding a better solution shortly. Intensification, conversely, refers to exploiting the current solution by focusing the search on a promising region of the search space. This will strictly improve the objective. It is essential to balance diversification and intensification to get the best possible final objective. An example of a framework that employs large neighborhoods is Adaptive Large Neighborhood Search (ALNS) (Pisinger and Ropke, 2007). ALNS uses destroy and repair pairs on the solution, guided by a dynamic selection mechanism that adapts to the search history.

2.2.3 Metaheuristics

A metaheuristic is a high-level problem-independent algorithmic framework that provides guidelines or strategies to develop heuristic optimization algorithms (Sörensen and Glover, 2013). Metaheuristics are problem-independent because they can be applied to various optimization problems without requiring specific knowledge about the problem or structure. This is because metaheuristics are based on general-purpose search and optimization techniques that can be applied to any optimization problem. While they are problem-independent, they still require a good understanding of the specific problem to design an appropriate search strategy and choose the algorithm's parameters and settings. Problem-specific heuristics can also help get better results. Metaheuristics aims to find a good, but only sometimes the optimal, solution in a reasonable time. They iteratively explore the solution space by making minor adjustments to the current solution and gradually improve it until a satisfactory solution is found. Some of the most widely used metaheuristics, such as Simulated Annealing (SA), Genetic Algorithms (GA), and ALNS, are designed for perturbative heuristics.

2.3 Hyperheuristics

A hyperheuristic is an automated methodology for selecting or generating heuristics to solve hard computational search problems (Burke et al., 2019). In combinatorial optimization, they have been defined as a heuristic to choose heuristics. Many hyperheuristics use high-level methodologies together with a set of low-level heuristics. Because of this, they are often problem-independent. Construction and perturbation can be used to refer to these low-level heuristics.

Hyperheuristics can be broadly classified into heuristic selection and heuristic generation (Drake et al., 2020). Heuristic selection works by selecting the most appropriate heuristic from a predefined set based on the current problem instance or situation. The selection is often guided by performance metrics or learning mechanisms that are aimed at maximizing the effectiveness of the chosen heuristic for the problem at hand. Heuristic generation generates new heuristics or modifies existing ones. This can be done through various methods, including genetic programming or machine learning techniques. The goal is to generate a heuristic well-suited to the current problem instance and potentially improve upon the performance of pre-existing heuristics. This thesis focuses on heuristic selection, and later I will elaborate on a deep reinforcement learning agent used for this purpose.

A hyperheuristic can be considered a learning algorithm when it uses some feedback from the search process. Not getting this feedback is called a non-learning hyperheuristic. We can categorize learning into two types: online and offline learning. Online learning occurs during the problem-solving process while the algorithm is running. Conversely, offline learning involves collecting rules or programs from a set of training instances, which hopefully generalize to solving unseen instances. A large part of this thesis is the Deep Reinforcement Learning Hyperheuristic (DRLH), which guides the selection of heuristics using deep reinforcement learning.

The main advantages of hyperheuristics include their generality (they can be applied to a wide variety of problems), simplicity (once the hyperheuristic is designed, it can be used with minimal adjustments), and adaptability (they can learn to improve over time). However, they may not always provide the optimal solution, especially for complex problems where a problem-specific heuristic might perform better.

2.4 Adaptive Large Neighborhood Search

Adaptive Large Neighborhood Search (ALNS) extends the Large Neighborhood Search (LNS) framework of Shaw (1998). The problem-solving approach is similar to the ruin-and-recreate principle by Schrimpf et al. (2000). ALNS iteratively destroys part of the solution and recreates it differently, hoping to find a better solution according to some criterion. In general, there are several destroy and several recreate heuristics. One destroy, and one recreate operation are chosen and applied to the current solution. In LNS, they are chosen randomly, while in ALNS, they are selected using adaptive weights.

In traditional LNS, destroy-recreate pairs make minor changes to the current solution. Ropke and Pisinger (2006) suggest using large moves that can rearrange up to 30-40% of the solution. They reason that minor changes might have difficulties moving from one promising area of the solution space to another. By making large changes to the *incumbent*, meaning current, ALNS can move to a new region of the search space far away from the incumbent solution, potentially leading to better solutions. In contrast, minor changes that only make small adjustments to the incumbent solution are more likely to keep the algorithm trapped in a local optimum. Another advantage of using big moves is that they can help to diversify the search by creating more diverse solutions. However, it is important to note that big moves must be balanced with local search methods, for example, Simulated Annealing (SA), that can refine the solutions and improve their quality.

2.4.1 Simulated Annealing

Simulated Annealing (SA) is a metaheuristic optimization algorithm inspired by the annealing process in metallurgy. It is commonly used to solve combinatorial optimization problems. SA begins with an initial solution and a temperature parameter. It iteratively explores the solution space by making small random changes to the current solution, similar to local search methods. What sets SA apart is its ability to accept worse solutions occasionally. This feature allows the algorithm to escape local optima and continue searching for potentially better solutions.

The acceptance of worse solutions in SA is determined by a probabilistic acceptance criterion (Kirkpatrick et al., 1983). This criterion considers the difference in objective function values between the incumbent and new solutions and the current temperature. Higher temperatures result in a higher probability of accepting worse solutions, enabling a broader exploration of the solution space. Therefore, the temperature gradually reduces as the algorithm progresses according to a predefined cooling schedule.

The cooling schedule is a critical factor in SA's performance. It controls the rate at which the temperature decreases over time. A well-designed cooling schedule strikes a balance between exploration and exploitation, gradually reducing the temperature to favor the selection of better solutions. Combining random moves with the ability to accept worse solutions at the beginning and progressively becoming more selective, SA can effectively explore the search space and converge towards an optimal or near-optimal solution.

2.4.2 Acceptance Criterion

The acceptance criterion is crucial to the ALNS algorithm because it determines whether a candidate solution should be accepted or rejected. The acceptance criterion plays an essential role in balancing the exploration of the search space and the exploitation of promising areas of the search space. Several acceptance criteria can be used in ALNS, each with advantages and disadvantages. The most straightforward acceptance criterion is to accept all candidate solutions. This approach can lead to rapid search space exploration but may result in poor-quality solutions.

Using the SA acceptance criterion in ALNS is common. This criterion states that a new solution should be accepted if it is better than the current solution or with a certain probability if it is worse. This probability is based on the Boltzmann probability function; see equation 2.1. The likelihood of accepting a worse solution is determined by a parameter called *temperature*, which gradually decreases over the course of the algorithm. This allows for exploration of the search space early in the algorithm, when the temperature is high, and exploitation of promising areas later in the algorithm, when the temperature is low. The rate at which the temperature decreases determines the balance between exploration and exploitation and can significantly impact the algorithm's performance.

The Boltzmann probability function used in SA. $f(s)$ is cost of the incumbent solution, $f(s')$ is the cost of the next solution, and T is the current temperature.

$$e^{-\frac{f(s)-f(s')}{T}} \tag{2.1}$$

2.4.3 Adaptive Weight Adjustment

ALNS uses adaptive weight adjustment to track how each heuristic performs. The entire search is divided into segments. At the start of each segment, each heuristic is given a score of zero. Then, each time a heuristic is used, points are added to the score to indicate how the heuristic performed. The idea behind this is to reward heuristics that find better solutions. However, diversification is essential to search a large part of the solution space. Therefore, heuristics that find a new, unseen solution are rewarded. The destroy and recreate operators are equally rewarded when used together. This is because it is hard to know which operator caused the success (Ropke and Pisinger, 2006). The adaptive weight adjustment helps ALNS balance intensification and diversification throughout the search, increasing its effectiveness in finding high-quality solutions to complex optimization problems.

The weight adjustment mechanism in ALNS is typically based on a reinforcement learning approach, where the weights of the heuristics are updated based on the reward signal received in each iteration. The reward signal is based on improving the objective value. Heuristics that lead to a better objective function value are given a higher reward, and their weights are increased accordingly. On the other hand, heuristics that lead to a worse objective function value are given a lower reward, and their weights are decreased accordingly.

Typically, there are three ways to gain a reward. Find a new global best solution. This usually gives the highest reward. Find a solution better than the incumbent solution. This usually gives a medium reward. Find a new, unseen solution. This usually gives the smallest reward. Otherwise, zero points are given as reward. These rewards are passed as parameters to ALNS, see Table 2.1.

Score	Description
$\sigma 1$	The last remove-insert operation resulted in a new global best solution. Usually gives the highest reward.
$\sigma 2$	The last remove-insert operation resulted in a solution that has not been accepted before. The cost of the new solution is better than the cost of the incumbent solution. Usually gives a medium reward.
$\sigma 3$	The last remove-insert operation resulted in a solution that has not been accepted before. The cost of the new solution is worse than the cost of the current solution, but the solution was accepted. Usually gives the lowest reward.

Table 2.1: Score parameters in ALNS. Table from Ropke and Pisinger (2006).

2.4.4 Parameter Tuning

ALNS involves various design choices and parameter settings Burke et al. (2013). These parameters govern the exploration and exploitation trade-off, the search strategies, and the balance between diversification and intensification. Different parameter values can lead to different search behaviors and can significantly impact the algorithm's performance, convergence rate, and solution quality.

Since ALNS aims to balance exploration and exploitation, determining the appropriate values for its parameters often requires experimentation and fine-tuning. These parameters may include acceptance criteria, perturbation strength, solution destruction and repair strategies, diversification mechanisms, intensification strategies, and neighborhood structures. By carefully adjusting these parameters based on problem characteristics, problem instances, and domain knowledge, practitioners can guide ALNS toward better performance and improve its ability to find high-quality solutions.

2.5 Reinforcement Learning

Machine Learning can be split into three subcategories: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, the learning algorithm is provided with labeled training data consisting of input-output pairs. The goal is to learn a mapping from inputs to outputs such that the model can make accurate predictions on new, unseen data. Supervised learning is generally used for tasks such as classification (e.g., identifying whether an email is spam) and regression (e.g., predicting house prices based on features). In unsupervised learning, the learning algorithm is provided with unlabeled data containing input features but no target outputs. The goal is to discover hidden patterns, relationships, or structures within the data. Unsupervised learning is used for tasks such as clustering (e.g., grouping customers based on purchase behavior) and dimensionality reduction (e.g., reducing the number of features while preserving important information).

2.5.1 Introduction to Reinforcement Learning

Reinforcement Learning is a type of machine learning that involves an agent interacting with an environment to learn how to make decisions. Unlike other machine learning approaches, reinforcement learning is based on trial and error. The agent acts in the environment, and the environment provides feedback through rewards or penalties. The agent aims to learn a *policy* that maximizes its cumulative reward, also called *return*. This approach is beneficial when there is no pre-existing labeled dataset, or the problem is too complex to be solved through traditional programming methods. Reinforcement learning has been used to solve a wide range of problems, from game playing and robotics to healthcare and finance.

Figure 2.1 shows how an agent interacts with an environment. At each time step t, the agent observes the current state S_t of the environment and selects an action A_t based on its current policy. The action taken by the agent causes the environment to transition to a new state $S_t + 1$, and the agent receives a reward $r_t + 1$. The reward signal indicates how well the agent performs at the current time step and is used to guide the agent's learning process. The agent's ultimate goal is to learn a policy that maximizes its return. Overall, figure 2.1 provides a high-level overview of the interaction between an agent and an environment in the context of reinforcement learning.

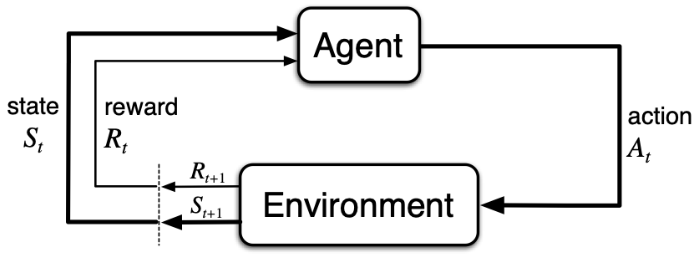


Figure 2.1: Agent and environment interaction. Figure from Sutton and Barto (2018).

2.5.2 Policy

A policy is a mapping from states to actions. It specifies what action an agent should take in a given state of the environment. The policy is the core component of a reinforcement learning algorithm because it determines the agent's behavior. There are two main types of policies in reinforcement learning: deterministic and stochastic. A deterministic policy specifies a single action for each state. Given a state s, the policy π(s) returns a single action a. On the other hand, a stochastic policy returns a probability distribution over actions for each state. Given a state s, the policy π(s) returns a probability distribution over the possible actions.

2.5.3 Reward function

A reward function is a function that maps a state-action pair to a numerical value that represents the desirability of taking that action in that state. The reward function is a critical component of a reinforcement learning algorithm because it guides the agent's behavior by providing feedback about the desirability of its actions. The reward function is specified by the designer of the reinforcement learning algorithm and depends on the task being solved. The reward function is typically designed to encourage the agent to take actions that lead to desirable outcomes and discourage actions that lead to undesirable results. For example, in a chess game, the reward function might give a positive reward for winning and a negative reward for losing the game.

The reward function can be designed to incentivize the agent to behave in a certain way. However, there is a risk that the agent may exploit the reward function to achieve its goals in unintended ways. For example, if the reward function for a cleaning robot is based on how clean the floor is, the robot may learn to spread dirt around to maximize its return. This phenomenon is known as reward hacking or reward engineering, and it is a significant challenge in reinforcement learning research (Hadfield-Menell et al., 2020).

Designing a good reward function is challenging because it requires carefully balancing the competing goals of encouraging the agent to explore the environment and exploiting the agent's current knowledge. If the reward function is too sparse or dense, the agent may become stuck in local optima or fail to explore the environment effectively. Therefore, designing a good reward function is often an iterative process that involves testing and refining different reward functions until the desired behavior is achieved.

2.5.4 Value functions

A value function is a function that estimates the expected return of being in a given state or taking a particular action in a given state. The value function is a critical component of many reinforcement learning algorithms because it provides a way to evaluate the quality of different policies and to update the policy iteratively to find the optimal one. There are two main types of value functions in reinforcement learning: state-value and action-value functions. A state-value function estimates the expected return of being in a given state and following a particular policy. It represents the value of being in a particular state, regardless of the action taken. The state-value function is denoted as V(s) and is defined as the expected return starting from state s and following policy π. An action-value function, on the other hand, estimates the expected return of taking a particular action in a given state and following a particular policy. It represents the value of taking a particular action in a particular state. The action-value function is denoted as Q(s, a) and is defined as the expected return starting from state s, taking action a, and following policy π.

State-value and action-value functions can be estimated using iterative methods, such as dynamic programming, Monte Carlo methods, and temporal-difference learning. These methods use different techniques to update the value function based on the observed rewards and state transitions.

The value function plays a crucial role in reinforcement learning because it provides a way to evaluate different policies and update the policy iteratively to find the optimal one. The optimal policy is the one that maximizes the expected return over time. Sometimes, the value function can derive the optimal policy directly without requiring an iterative search.

2.5.5 Approaches

There are several approaches to finding an optimal policy that maximizes the expected return. The three most common approaches are value-based, policy-based, and actor-critic methods (Sutton and Barto, 2018).

Value-based methods learn the value function of each state or state-action pair and use it to derive an optimal policy. The most common value-based method is Q-learning, which learns the action-value function directly. Q-learning updates the Q-values iteratively based on the observed rewards and state transitions and uses a greedy policy to choose the action that maximizes the Q-value for a given state. Value-based methods are beneficial when the action space is significant, as they can scale well to high-dimensional problems.

Policy-based methods learn the policy directly without explicitly estimating the value function. The most common policy-based method is the policy gradient method, which iteratively updates the policy in the direction of the gradient of the expected return with respect to the policy parameters. Policy-based methods are beneficial when the action space is small and discrete, as they can handle both deterministic and stochastic policies.

Actor-Critic methods combine the advantages of value-based and policy-based methods by learning the value function and the policy. Actor-critic methods consist of two components: an actor that learns the policy and a critic that learns the value function. The critic provides feedback to the actor about the quality of its policy, and the actor updates the policy based on this feedback. Actor-critic methods are beneficial when the action space is large and continuous, as they can handle both deterministic and stochastic policies.

2.6 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) combines deep neural networks with reinforcement learning to enable artificial agents to learn complex behaviors and decision-making skills in dynamic environments. Deep neural networks are often used as function approximators to represent the value function or policy. By using deep neural networks to approximate value functions and policies, DRL can address problems with large or continuous state and action spaces, making it applicable to a wide range of complex tasks.

Value function approximation: Instead of maintaining a lookup table for value functions, which is infeasible for high-dimensional state spaces, deep neural networks are used to approximate the value functions. Given an input state or state-action pair, the network learns to output the corresponding estimated value. The most common architecture for approximating the action-value function is the Deep Q-Network (DQN), which combines Q-learning with deep learning.

Policy approximation: In policy-based DRL methods, a deep neural network represents the policy directly. The input to the network is the state, and the output represents the probability distribution over possible actions or the deterministic action to be taken. Policy gradient methods, like REINFORCE, and advanced algorithms, such as Proximal Policy Optimization (PPO) and Actor-Critic Methods (ACM), are commonly used for policy approximation. Policy gradient methods aim to maximize the expected return by directly optimizing the policy parameters. First, the gradient of the objective function is estimated using the log probability of the chosen action and the return (or advantage function). The policy parameters are then updated using gradient ascent.

2.6.1 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a policy gradient-based reinforcement learning algorithm developed by OpenAI. It aims to address the challenges of sample efficiency and stability in training by maintaining a balance between exploration and exploitation. PPO improves the stability of the learning process by ensuring that the updated policy does not deviate too far from the old policy. This is achieved using a trust region optimization technique, which constrains the updates to the policy. PPO has been successfully applied to various tasks, including robotics, game playing, and continuous control problems.

The main idea behind PPO is to update the policy by taking multiple steps using the same set of collected trajectories (i.e., state, action, and reward sequences) while ensuring that the policy does not change too drastically. To achieve this, PPO introduces a clipped function that penalizes the policy update if the new policy moves too far from the old policy (Schulman et al., 2017).

Another critical aspect of PPO is using a value function to estimate the expected return of a given state or action. By learning both a policy and value function simultaneously, PPO can leverage both sources of information to improve performance. Additionally, PPO uses a trust region approach to ensure that policy updates stay within a specific range, allowing the algorithm to achieve better sample efficiency and convergence compared to other reinforcement learning methods (Schulman et al., 2017).

While PPO has been successful in various domains, the algorithm still has some limitations. One of the main challenges is the selection of hyperparameters, which can significantly impact the algorithm's performance (Schulman et al., 2017). Additionally, PPO may struggle with tasks that require long-term planning or complex reasoning, as the algorithm relies on a relatively simple policy function that may not be able to capture the nuances of these tasks.

2.7 Related work

2.7.1 Vehicle Routing Problem with Autonomous Assistants

Chen et al. (2021) solved a Vehicle Routing Problem with Time Windows and Delivery Robots (VRPTWDR) using ALNS. They found high-quality solutions on up to 200-node instances. Furthermore, they found that adding delivery robots decreases the objective function dramatically. Moshref-Javadi et al. (2020) found similar results when applying an ALNS inspired algorithm on different versions of the Traveling Repairman Problem with Drones. They reported significant improvement when using drones in combination with a truck compared to using only the truck. Ottomanelli et al. (2017) also used a heuristic approach and were able to beat baseline instances of Traveling Salesman Problem (TSP) when introducing drones. Although there are some differences between drones and robots, the conclusion remains the same. Using autonomous delivery assistants for last-mile delivery can significantly improve solutions for various Vehicle Routing Problems.

2.7.2 Hyperheuristics

Zhang et al. (2022) proposed a deep reinforcement-based hyperheuristic framework to deal with uncertainties encountered in real-life problems. The proposed approach used a Double Deep Q-learning network. It was assessed on two combinatorial optimization problems, a real-world container terminal truck routing problem with uncertain service times and the online 2D strip packing problem. Zhang et al. (2022) used manual heuristics for baseline performance evaluations. However, they noted that manual heuristics for performance evaluations are often far from optimal. Their results demonstrated superior performance compared to existing solution methods for these problems.

Lu et al. (2020) proposed using a Deep Reinforcement Learning (DRL) agent to select the optimal low-level heuristic at each stage when dealing with the Capacitated Vehicle Routing Problem (CVRP). Their approach struggled with transferability to other optimization challenges because the components of the DRL agent were uniquely tailored to the CVRP. In their design, the agent's training was primarily geared towards intensification, sidelining diversification. A rule-based escape method was enforced instead of allowing the RL agent to find a balance between intensification and diversification. Letting the agent learn this balance could potentially lead to better results. Kallestad et al. (2023) introduced Deep Reinforcement Learning Hyperheuristic (DRLH), a general selection hyperheuristic framework that can generalize to various optimization problems. This framework can be read about in Chapter 4.

Chapter 3

Flexible Traveling Repairman Problem with Drones

The Flexible Traveling Repairman Problem with Drones (FTRPD) is an extension of the Traveling Repairman Problem (TRP). TRP, also known as the Minimum Latency Problem, is a basic routing problem that determines the sequence of visits of a single vehicle to a given set of customer locations in an attempt to minimize the sum of waiting times of all customers (Moshref-Javadi et al., 2020). In the FTRPD, the truck is combined with multiple Unmanned Aerial Vehicle (UAV) operations. The truck serves as a mobile depot for the UAVs, also called drones. Deliveries can be made by both truck and drone. While the truck follows a multi-stop route, each drone delivers a single shipment per dispatch. The apparent advantage is that UAVs can travel flexibly in three dimensions. This makes delivering easier since a discrete set of static roadways does not restrict the UAVs.

UAVs suffer from two significant limitations. First, today's battery technology limits the flight time of a UAV and, in turn, its geographical reach. In recent surveys, the available flight time for freight-bearing UAVs ranges from 30 to 40 minutes (Kolodny, 2016; Lardinois, 2016; Levine, 2016). Second, their physical carrying capacity is limited regarding package size and weight. Typically, UAVs used for urban logistics are limited to one package per dispatch. The FTRPD considers these two limitations by introducing a *drone limit* and restricting drones to one package per dispatch.

The *drone limit* signifies the maximum distance a drone can travel before needing a recharge, and its influenced by the *coverage percentage*. This percentage indicates the proportion of feasible unique drone routes. A drone route involves three steps: launching from the *launch node*, delivering to the *delivery node*, and returning to the *reconvene node*. The total travel time of a route must be lower than or equal to the drone limit for a route to be feasible. A coverage percentage of 25% means that 25% of all unique drone routes in an instance should be feasible. A higher coverage percentage will mean a higher drone limit since it implies that a larger fraction of unique drone routes needs to be feasible.

A UAV can deliver one package per dispatch and must reconvene with the truck before picking up another package. Likewise, drones must be launched and reconvened at customer locations. In the FTRPD, drones can be reconvened at the node they are launched from. However, if a customer location is a reconvene node, the truck must wait for the drone(s) to arrive before it can drive to the next customer node. Likewise, the UAVs must wait for the truck if they arrive at a reconvene node first. This waiting time is included in the total travel time of a drone route. Meaning, that if the waiting time makes the total travel time exceed the drone limit, this delivery will be infeasible.

3.1 Truck and Drone Routing Algorithm

Truck and Drone Routing Algorithm (TDRA) was proposed by Moshref-Javadi et al. (2020) to solve different versions of the Traveling Repairman Problem with Drones (TRPD). It is an extension of ALNS Pisinger and Ropke (2007). The main difference is using a Shake heuristic after $\xi * c$ number of non-improving iterations. See Algorithm 1 for TDRA pseudocode and Table 3.1 for an explanation of the parameters.

Algorithm 1 TDRA Pseudocode (Moshref-Javadi et al., 2020)

```
Inputs: $(d_{ij}, \hat{d}_{ij}, U, C_0, c, \xi, R_{\text{max}}, T_0, \beta_{\text{TDRA}})$
Generate an initial solution, $s$, to the FTRPD based on a Simulated Annealing approach
$s_{\text{best}} \leftarrow s, f(s_{\text{best}}) \leftarrow f(s), iter \leftarrow 0, T \leftarrow T_0$
while $iter \neq R_{\text{max}}$ do
    Select a heuristic from the list of heuristics using the selection method
    Apply the selected heuristic to $s'$
    Save the new solution as $s'$
    if $f(s') < f(s_{\text{best}})$ then
        $s_{\text{best}} \leftarrow s', f(s_{\text{best}}) \leftarrow f(s')$
    end if
    if $s'$ is accepted as the new solution based on the Boltzmann probability function with current temperature $T$ then
        $s \leftarrow s'$
    end if
    Update weights of heuristics
    Change search neighborhood using Shake heuristic after $\xi c$ non-improving iterations
    $iter \leftarrow iter + 1$
    $T \leftarrow \beta_{\text{TDRA}} T$
end while
return $s_{\text{best}}$, $f(s_{\text{best}})$
```

Parameter	**Description**
d_{ij}	Travel time between node i and j by truck.
$\hat{d}_{ij}$	Travel time between node i and j by UAV.
U	Set of UAVs.
C_0	Set of customers and the depot as starting location. $C_0 = 0, 1, 2, ..., c$.
c	Number of customers.
ξ	Shake heuristic coefficient.
R_{max}	Maximum number of iterations.
T_0	Initial temperature.
β_{TDRA}	Temperature coefficient.

Table 3.1: Explanation of TDRA parameters.

3.2 Initial solution

TDRA begins with an initial feasible solution. This solution is generated by assigning all customers randomly to the truck route. Afterward, a Simulated Annealing (SA) algorithm improves the generated solution. The SA starts with an initial temperature T_0 and is decreased by factor β_{SA} each iteration. The SA acceptance criterion is used as the acceptance function. The SA algorithm uses two-opt and three-opt to find a better solution. The heuristics are only used on the truck route, meaning the output is a TRP solution since all customers are assigned to the truck. See Figure 3.1 for an example TRP solution.

The output of SA is then given to the Elliptical Customer Assignment (ECA) heuristic. ECA assigns some customers from the TRP solution to the drones. The Least-Squares criterion estimates the best ellipse fit to the given set of customers and depot locations. Next, ECA tries to assign the customers furthest away from the ellipse to the drones. The distance is calculated using the Euclidean distance between the ellipse and the customer node. See algorithm 2 for pseudocode.

Algorithm 2 ECA Pseudocode

```
Inputs: $s_{in} \leftarrow$ TRP solution from SA, $C_0$
$s_{out} \leftarrow s_{in}, f(s_{out}) \leftarrow f(s_{in})$
Fit an ellipse to $C_0$ according to Gander et al. (1994)
Make a List of all customers, sorted in descending order of their distance from the ellipse
while No feasible position is available for customers in the List for re-insertion in UAV routes
do
    j $\leftarrow$ index of $i^{th}$ customer in the List
    Select the customer $c_j$ in the List and remove it from the truck route
    for all potential positions in all UAV and truck routes in $s_{out}$ to insert customer $c_j$ do
        Re-insert customer $c_j$ in the current position and save the new solution as s'
        if (s' is feasible) and (f(s') ¡ f($s_{out}$) then
            $s_{out} \leftarrow$ s'
            $f(s_{out}) \leftarrow$ f(s')
        end if
    end for
    Remove the assigned customers from the List
end while
return $s_{\text{best}}, f(s_{\text{best}})$
```

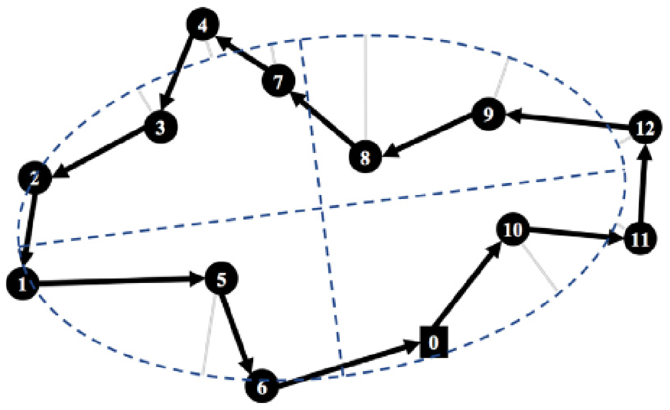


Figure 3.1: Solution Representation before the ECA heuristic is applied. See figure 3.2 for a solution representation where the ECA heuristic has been applied. Figure from Moshref-Javadi et al. (2020)

3.3 Solution Representation

The solution representation is made up of 4 parts. Part 1 is the truck route. The truck starts and stops at the depot, which is noted as 0. Part 2 denotes customers assigned to UAVs and the sequence of visits. "X" differentiates the UAV routes. Parts 3 and 4 represent the launch and reconvene locations for each UAV. Parts 3 and 4 refer to cell numbers in Part 1. For instance, number 1 in Part 3 means a drone is launched from the depot because the depot is the first cell in Part 1. Figure 3.2 is an example of a solution representation with ten customers and two UAVs.

3.4 Objective

The *MinSum* objective, also known as the minimum sum objective, is used to determine the cost of a solution. The goal is to minimize the waiting times of all customers. Many retailers compete for consumer demand and loyalty, and as noted by Jacobs et al. (2019), fast and frequent deliveries have emerged as a crucial factor for differentiation. Therefore, the MinSum objective is a logical choice for the FTRPD.

3.5 Heuristics

This is the heuristic set used by Moshref-Javadi et al. (2020). It is a combination of general- and problem-specific heuristics. There are general operators such as Two-opt

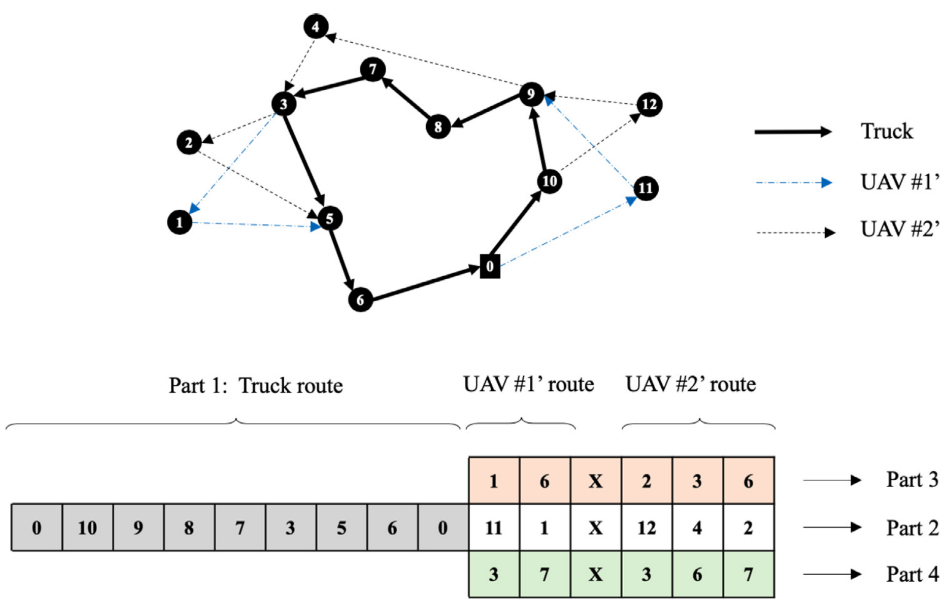


Figure 3.2: Solution Representation of the FTRPD. Figure from Moshref-Javadi et al. (2020)

.

and Three-opt. Furthermore, general principles such as greedy- and random re-insert are used. However, these principles have been combined with problem-specific knowledge. E.g., Origin-destination relocation greedily re-inserts a customer used as a rendezvous location. Finally, some operators are problem-specific, such as the Drone Planner.

Two-opt

This operator considers parts 1 and 2 of the solution representation as a single vector. It selects two random customers and swaps them. If the solution is infeasible, the drone planner heuristic is applied in hopes of finding a feasible solution. The original solution is returned if the operator cannot find a new, feasible solution.

Three-opt

Three-opt works similarly to two-opt. It considers parts 1 and 2 of the solution representation as a single vector. It selects three random customers and swaps them. If the solution is infeasible, the drone planner heuristic is applied in hopes of finding a feasible solution. The original solution is returned if the operator cannot find a new, feasible solution.

Greedy assignment

Greedy assignment selects a random customer, not used as a rendezvous location, from parts 1 or 2 of the solution representation. A rendezvous location means a drone is launched from or reconvened at this customer. The reason rendezvous locations are not considered is because special operators are used for rendezvous locations. Then, the customer is removed from the solution and re-inserted at the most advantageous position. All truck and drone positions are considered, including all combinations of launch and reconvene locations. The original solution is returned if the operator cannot find a new, feasible solution.

Origin-destination relocation

This heuristic complements the greedy assignment heuristic by considering customers used as rendezvous locations. A random customer is selected, removed from the solution, and re-inserted at the most advantageous position. Switching around rendezvous locations may result in infeasible solutions. Therefore, the drone planner heuristic is applied to ensure a higher possibility of finding a feasible solution. The original solution is returned if the operator cannot find a new, feasible solution.

General Assignment

This heuristic works similarly to the greedy assignment heuristic. However, it selects 2-5 random customers not used as rendezvous locations. Then, each customer is removed from the solution and re-inserted at the most advantageous position. The original solution is returned if the heuristic cannot find a feasible solution with all customers re-inserted.

Drone Planner

The drone planner heuristic considers parts 3 and 4 of the solution representation. It goes through every customer assigned to drones and tries to find a launch and reconvene pair that gives a better objective. Note that since part 2 of the solution remains unchanged, the assignment of customers to UAVs remains unchanged. The original solution is returned if the heuristic cannot find a feasible solution with all customers re-inserted.

Wild change

Wild change selects 2 to 5 customers and randomly re-inserts them in the truck or drone. This may cause the new solution to be infeasible. Therefore, the drone planner heuristic is used to increase the chances of finding a feasible solution. The original solution is returned if the heuristic cannot find a feasible solution with all customers re-inserted.

3.6 Acceptance Criterion

The SA acceptance criterion, introduced in Kirkpatrick et al. (1983), is used as the acceptance criterion. A new solution is accepted if it is better than the incumbent solution. Otherwise, a temperature parameter and the Boltzmann probability function decide if a solution should be accepted. The temperature decreases throughout the search. This allows for exploring an extensive range of potential solutions at the start of a search and exploiting the most promising neighborhoods during the final stages of a search.

3.7 Heuristic Selection and Adaptive Weight Adjustment

Heuristics are chosen based on the roulette wheel. Each heuristic has a probability of being chosen. Initially, all heuristics have an equal chance of being selected. The entire search comprises $R_{\max}$ iterations. These iterations are divided into several segments. During a segment, the points and number of uses are tracked for each heuristic. 2 points are gained when improving the global best solution, and 1 point is achieved when improving the incumbent solution. The weights are updated at the end of each segment. The heuristics that performed well in the last segment will more likely be selected in the next segment. The hyperparameter $\gamma \in [0.0, 1.0]$ decides how much the weights are adjusted. It is essential to balance γ so the TDRA uses the best heuristics in each search segment. Heuristics that do well in a specific segment should be used more. However, heuristics that have performed well throughout the search should also be used.

3.8 Shake heuristic

The Shake Heuristic is used after $\xi * c$ number of *non-improving iterations.* ξ is a tunable parameter that decides when the shake heuristic is applied, and c is the number of customers. The shake heuristic should not be used too early because the operators should be given a chance to find a better solution. However, the shake heuristic should not be applied too late because many iterations might be wasted due to being stuck in a local minimum. The shake heuristic is designed to drastically change the solution search space to escape local minima. General Assignment and Wild Change are used in the shake heuristic since these operators are best at diversifying (Moshref-Javadi et al., 2020). These are applied to the incumbent solution n times. n is a tunable parameter. This parameter should be high enough so that the shake heuristic can escape local minima. However, it should not be too high since shaking too much could lead to exploring less promising areas of the solution space.

Chapter 4

Deep Reinforcement Learning Hyperheuristic

Deep Reinforcement Learning Hyperheuristic (DRLH) was proposed by Kallestad et al. (2023) as a general framework for solving combinatorial optimization problems. It is a hyperheuristic approach based on Deep Reinforcement Learning (DRL) and ALNS. DRLH uses a Reinforcement Learning agent to select heuristics. Kallestad et al. (2023) showed that DRLH performed better than ALNS on several routing problems, including Capacitated Vehicle Routing Problem (CVRP), Parallel Job Scheduling Problem (PJSP), Pickup and Delivery Problem (PDP), and Pickup and Delivery Problem with Time Windows (PDPTW).

DRLH is proposed as a general selection hyperheuristic framework for solving combinatorial optimization problems. It uses problem-independent state space, heuristics, and reward function. This makes it adaptable to many different problems. Later in the thesis, I will compare TDRA and DRLH on the FTRPD.

4.1 State Space

The state space consists of 13 unique features. General features have been prioritized so that the state space can be used for many combinatorial optimization problems. Table 4.1 lists all state features and their descriptions.

Name	Description
reduced_cost	The difference of cost between previous & current solutions.
cost_from_min	The difference of cost between current & best_found solutions.
cost	The cost of the current solution.
min_cost	The cost of the best-found solution.
temp	The current temperature.
cs	The cooling schedule (α).
no_improvement	The number of iterations since the last improvement.
index_step	Iteration number.
was_changed	1 if the solution was changed from the previous, 0 otherwise.
unseen	1 if the solution has not previously been encountered in the search, 0 otherwise.
last_action_sign	1 if the previous step resulted in a better solution, 0 otherwise.
last_action	The action in the previous iteration encoded in 1-hot.

Table 4.1: State space in DRLH. Table from Kallestad et al. (2023).

4.2 Reward function

DRLH uses the reward function called R_t^{5310}. This reward function rewards a new global best solution with 5 points. A better solution than the incumbent solution is rewarded with 3 points, a new unseen solution is given 1 point, and 0 points are given otherwise. R_t^{5310} takes inspiration from the original reward function used in the adaptive weight adjustment in ALNS. The goal is to reward the agent when it finds better solutions. The 1 point from new unseen solutions is there to encourage diversification. It is important to note that the Proximal Policy Optimization (PPO) algorithm considers future rewards, not only immediate rewards when training the agent. This means that DRLH indirectly encourages diversification because the agent can learn to take actions that give a smaller immediate reward in return for a larger reward in the future.

4.3 Acceptance Criterion

The SA acceptance criterion is used as the acceptance criterion. This acceptance criterion will accept a new solution, provided it outperforms the incumbent one according to a specific objective function. Otherwise, the Boltzmann probability function is used to decide if a solution should be accepted. The difference between the incumbent solution l and the new solution l' can be denoted $\Delta E = f(l') - f(l)$. With temperature parameter T, the new solution is accepted with probability $e^{-|\Delta E|/T}$.

4.4 Heuristics

Each heuristic $h \in H$ in DRLH combines a removal and an insertion operator. These are also known as destroy-repair operators. One removes a call from the solution, and the other inserts the call into a different place. There is also an additional operator used called "Find_single_best". This operator goes through every element in the current solution, removes it, and finds the best place to reinsert it. Finally, the reinsert that achieves the minimum cost is selected as the new solution. The operators are problem-independent and can be used for many combinatorial optimization problems (Kallestad et al., 2023).

4.5 Heuristic Selection

DRLH replaces the adaptive layer of ALNS with a deep reinforcement learning agent trained using PPO. The DRLH training process enables it to adapt to various conditions and settings and learn effective heuristic selection strategies. ALNS splits its search into multiple segments, meaning it makes decisions at a macro-level. DRLH is designed to make micro-level decisions, utilizing current search state information during heuristic selection. It can adjust its heuristic selection probabilities to the current search state information. Consequently, it can adapt to new search state information as soon as it becomes available.

In the usual Reinforcement Learning (RL) scenario, an agent is trained to perfect a policy π. This policy guides the agent in selecting actions based on its interaction with the environment. During every timestep t, the agent picks an action A_t and gets a numerical reward R_t from the environment. This reward serves as a measure of the action's effectiveness. The state S_t is information the agent gathers from the environment at each timestep, dependent on its selected action A_t from a list of potential actions. The agent's stochastic policy π is therefore represented as

$$\pi(a|s) = \Pr\{A_t = a|S_t = s\}. \tag{4.1}$$

One particular policy type is a parameterized stochastic policy function. In this function, the likelihood of choosing an action is influenced by a set of parameters $\theta \in R^d$ space. Hence, the earlier equation is reformulated as

$$\pi(a|s, \theta) = \Pr\{A_t = a|S_t = s, \theta_t = \theta\}, \tag{4.2}$$

where θ_t stands for the parameters at timestep t (Sutton and Barto, 2018).

In DRLH, the policy π is realized through a MultiLayer perceptron (MLP), a type of nonlinear function approximation (Goodfellow et al., 2016). This situation aims to approximate the optimal policy π^* by adjusting θ, symbolizing the weights of the MLP network.

Training the DRL agent

The training process for the DRL agent is outlined in Algorithm 3. To train the weights of the MLP, the policy gradient method of Proximal Policy Optimization (PPO), introduced by Schulman et al. (2017), is followed. To handle different variations of an optimization problem, the training process involves multiple problem instances or episodes, where each instance corresponds to a unique set of problem attributes. Each instance is optimized for a certain number of iterations or time steps. At the end of each episode, the policy parameters θ are updated until the optimal policy is obtained. Once the training process is completed, the optimal policy π^* is used to solve unseen instances in the test sets.

Algorithm 3 Training the Deep RL agent. Algorithm from Kallestad et al. (2023).

```
Result: $\pi*$ optimal policy
Start with random setting of $\theta$ for a random policy $\pi$;
for $e \leftarrow 1$ to episodes do
    Receive initial State $S_1$;
    for $t \leftarrow 1$ to steps do
        Choose and perform action $a \in A_t$ according to $\pi(a|s, \theta)$;
        Receive $R_t = v$ and $s \in S_(t + 1)$ from the environment
    end for
    Update the policy parameters $\theta$ according to PPO Schulman et al. (2017)
end for
```

4.6 Hyperparameter selection

The hyperparameters of DRLH play a crucial role in determining the model's training speed, stability, and final performance (Kallestad et al., 2023). A lower learning rate leads to longer training times but increases the likelihood of achieving better performance once the model is fully trained. Kallestad et al. (2023) used the same hyperparameters for four combinatorial optimization problems. Since the hyperparameters for DRLH are related to the high-level problem of heuristic selection, which stays the same regardless of the underlying combinatorial optimization problem, Kallestad et al. (2023) speculate that the selected hyperparameters will work for any underlying combinatorial optimization problem.

Chapter 5

Experimental Setup

Our goal is to compare the performance of TDRA and DRLH on the FTRPD. We are most interested in how the different heuristic selection strategies affect performance. Therefore, both TDRA and DRLH will use 1000 steps to solve each instance, the SA acceptance criterion is used as the acceptance criterion for both methods, and both methods will use the original set of heuristics from Moshref-Javadi et al. (2020). These steps will contribute to a fair comparison of heuristic selection.

All experiments have been run on an AMD Ryzen 5 2600X Six-Core Processor. DRLH has been trained on one NVIDIA A100 80GB GPU and an AMD EPYC 7742 64-Core Processor.

5.1 Baseline Sets

Both sets are in a 100x100 grid. The depot is randomly located within the grid. The baseline parameters for this thesis are established with the drone speed being 1.5 times faster than that of the truck and the drone's coverage to 25%. Remember, this is not 25% coverage of the total area but 25% coverage of all unique drone routes for an instance.

5.1.1 Set 3

Set 3, as referenced in Moshref-Javadi et al. (2020), is one of the chosen datasets. This is primarily because it contains 20 instances of sizes 10, 20, 50, and 100. As demonstrated by Kallestad et al. (2023), the effectiveness of DRLH over ALNS tends to increase as the problem size increases. I want to see if the same is true for the FTRPD. Therefore, set 3 is a logical choice because it contains small and large problem sizes.

5.1.2 Test Set

The test set contains 50 instances for sizes 10, 20, and 50. All coordinates have been randomly generated within a 100x100 grid. The depot has also been randomly generated.

5.2 Initial solution

I've chosen to omit the use of SA and ECA when comparing the performance of TDRA and DRLH. Instead, the initial solution is a TRP solution where customers are visited in sequential order, i.e., customer one is visited first, customer two is visited second, etc. This approach is motivated by three key reasons.

Firstly, TDRA and DRLH should start from the same solution to fairly compare the two methods. Using SA and ECA to find an initial solution will not guarantee that the same solution is given to both methods. If the ECA heuristic was used alone to find an initial solution, this problem would be mitigated. However, training a size 50 DRLH model with the ECA heuristic takes too long.

Secondly, the ECA heuristic is computationally expensive, especially for larger problem sizes. There are two significant bottlenecks during the training of DRLH models: ECA and the heuristic set. The heuristic set is an essential part of the experiments, and in my experience, the ECA heuristic is the biggest bottleneck. Therefore, it makes sense to omit it so that size 50 DRLH models can be trained in a reasonable time.

Thirdly, DRLH models have been trained to handle scenarios where it starts with a TRP solution. Therefore, testing it in scenarios where it is given a TRP solution makes sense. Furthermore, to fairly compare performance, TDRA should start from the same TRP solutions.

5.3 TDRA

I implemented the FTRPD and TDRA in Python 3.10 according to the pseudocode and logic by Moshref-Javadi et al. (2020). I used the same solution representation, heuristic set, and heuristic selection method. Furthermore, I decided to use the MinSum objective to measure the cost of a solution. The Euclidean distance between node i and node j is the truck travel time between these nodes. This also means that the first arrival at node j is the travel time between node i and node j plus the potential travel time before arriving at node i. The drone travel time between customer nodes depends on the *drone speed*. Truck and drone service times are set to zero. This means that packages are immediately delivered when the truck or drone arrives at a customer node, and they are immediately ready to depart. It is also important to note that drones are immediately recharged when returning to the truck.

5.3.1 Parameter Selection

As noted earlier in the thesis, parameter tuning is essential to balance exploration and exploitation. I have chosen to use the same score system as Moshref-Javadi et al. (2020). Furthermore, I have decided to use the same adaptive weight coefficient γ. Given that TDRA and DRLH will execute for a total of 1000 steps to seek the optimum solution, I have set R_{max}, the maximum iteration parameter, to be 1000. I found inspiration in the selected parameters of Moshref-Javadi et al. (2020), then tuned them for $R_{max} = 1000$ in initial experiments. See Table 5.1 for parameters, explanations, and selected values.

Parameter	**Explanation**	**Value**
R_{max}	Max number of iterations.	1000
Segment size	Iterations in each segment.	25
T_0	Initial Temperature.	800
β_{TDRA}	Temperature Cooling Coefficient.	0.995
η	Number of diversification operators applied in the Shake heuristic.	20
ξ	Shake Heuristic Coefficient.	5
High score	Given when a new global best solution is found.	2
Medium score	Given when the incumbent solution is improved.	1
Low score	Given otherwise.	0
γ	Adaptive weight coefficient. Used to balance new and old information when updating the weights.	0.2

Table 5.1: TDRA parameters in experiments.

5.4 DRLH

I have used the implementation of Kallestad et al. (2023). One of the main benefits of DRLH is that it can generalize to many combinatorial optimization problems. This is due to using a general state representation and a general reward function that can be used for many problems. Therefore, I have chosen to use the same state representation as Kallestad et al. (2023). Furthermore, I have used the same hyperparameters except for the learning rate. Table 5.2 shows hyperparameter values.

Hyperparameter	**Value**
Max epochs	5000
Learning rate	2.5e-5
Batch size	64
First hidden layer size	256
Second hidden layer size	256
Discount factor	0.5

Table 5.2: Hyperparameters used in DRLH. Table from Kallestad et al. (2023). Note that the learning rate is different from the original table. I explain this choice in Section 5.4.3.

5.4.1 Reward Functions

$R_t^{new\_best}$

I chose the reward function $R_t^{new\_best}$ for problem sizes 10 and 20. This gives a reward whenever the best global solution is improved. The reward signal equals how much the new global best solution improved upon the last global best solution. E.g., if the Min-Sum objective of the new global best is 100 lower than the previous global best, then the agent is rewarded 100 points. I chose this function because it aligned well with my goal of minimizing the objective and performed best during initial separate experiments. I experimented with different reward functions in which I observed a misalignment between my goal and the agent's goal of maximizing its return. You can read about the experiments in Appendix A.

$R_t^{new\_best/100}$

I chose the reward function $R_t^{new\_best/100}$ for size 50. This reward function works the same as $R_t^{new\_best}$, but the reward is divided by 100. It performed better than $R_t^{new\_best}$ in initial experiments for size 50. Engstrom et al. (2020) reported that reward scaling can significantly affect the performance of PPO. This is probably because rewards that are too large or too small can lead to numerical instability in the learning algorithm. For example, huge rewards can cause the gradients of the policy network to explode, while small rewards can make learning slow or even stall completely. This could be why $R_t^{new\_best/100}$ performs better than $R_t^{new\_best}$ on size 50. The rewards become too large. This was not the case for sizes 10 and 20, which would explain why $R_t^{new\_best}$ performed best in these cases.

5.4.2 Entropy Coefficient

In my implementation of PPO, I have incorporated an entropy coefficient into the overall loss calculation. The entropy of a probability distribution quantifies its uncertainty, and in the context of reinforcement learning, it describes the randomness of the policy's action selection. The entropy coefficient effectively serves as a regularization term, discouraging the policy from becoming deterministic too quickly (Espeholt et al., 2018).

By introducing this term, the model is urged to maintain a level of stochasticity in its action probabilities. This will encourage exploration, as it prevents the policy from focusing narrowly on currently known rewarding states, thereby allowing it to discover more beneficial actions potentially (Ahmed et al., 2019).

5.4.3 Learning Rate

I chose to change the learning rate because the learning rate often tends to have a more direct and substantial impact on the speed and the quality of learning than other hyper-parameters. The learning rate determines how much the policy is updated in response to each new batch of experience. Too high a learning rate can lead to unstable learning or overshooting the optimum, while too low a learning rate can cause slow convergence or getting stuck in suboptimal policies. Furthermore, tuning the learning rate effectively can help balance exploration and exploitation, speed up the rate of learning, and enhance stability and convergence. A benefit of DRLH is its generalizability. Therefore, using the same hyperparameters across different combinatorial optimization problems would be a massive benefit. This would lead to a plug-and-play framework that is easy to use. Because of this, I want to change as few hyperparameters as possible to see how they perform on the FTRPD.

I did some separate initial experiments to determine the learning rate. I chose $2.5e^{-5}$ because it performed best. I have used the same hyperparameters when training all models.

5.4.4 Dataset Generation

I generate a distinct training set of 5000 instances of the FTRPD. Instances of sizes $N = 10$, $N = 20$, and $N = 50$ are generated. All customers are randomly generated within a 100x100 grid. The depot is also randomly generated. All DRLH models will be trained with baseline parameters of drone speed = 1.5 and coverage percentage = 25 %.

Chapter 6

Results

6.1 Results on Set 3

Figure 6.1 shows the average improvement of DRLH over TDRA on Set 3. All instances have been run 10 times. We see that DRLH outperforms TDRA for all the instance sizes. Furthermore, we see that DRLH becomes increasingly effective as the problem size increases. This is the same pattern Kallestad et al. (2023) observed when introducing DRLH. Detailed numerical results on Set 3 are in Appendix B.1.3.

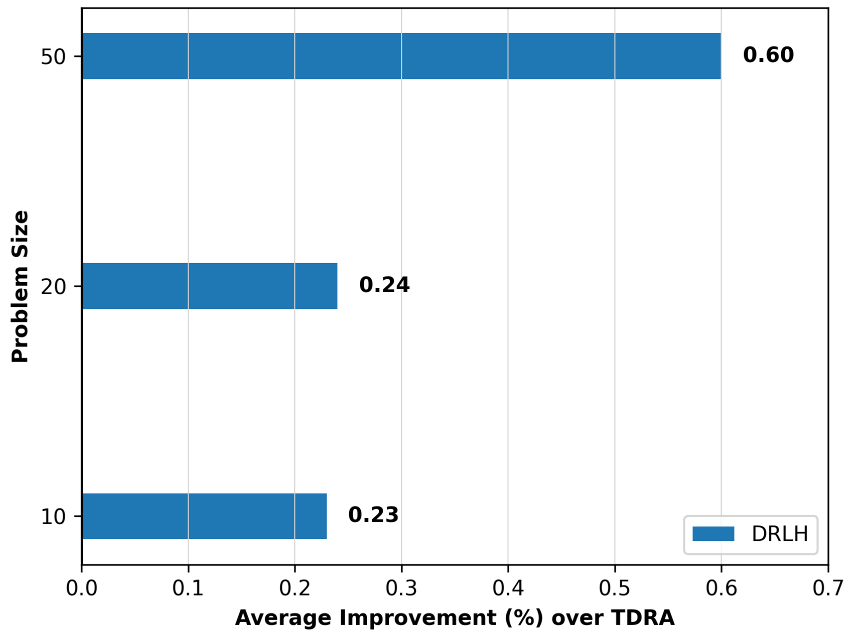


Figure 6.1: Results of DRLH on Set 3 using TDRA as baseline.

Figure 6.2 is a boxplot showing a comparison of the distribution of objective values obtained by TDRA and DRLH on Set 3. All instances have been run 10 times. The central lines represent the median values, the box boundaries indicate the interquartile range (25th and 75th percentiles), and the whiskers extend to the minimum and maximum values, excluding outliers. We observe similar performance for size 10. For sizes 20 and 50 TDRA exhibits higher variance, occasionally reaching superior objective values. Still, it generally performs less consistently. DRLH achieves a better median value for size 50, demonstrating its reliable performance, despite its lower best-case results.

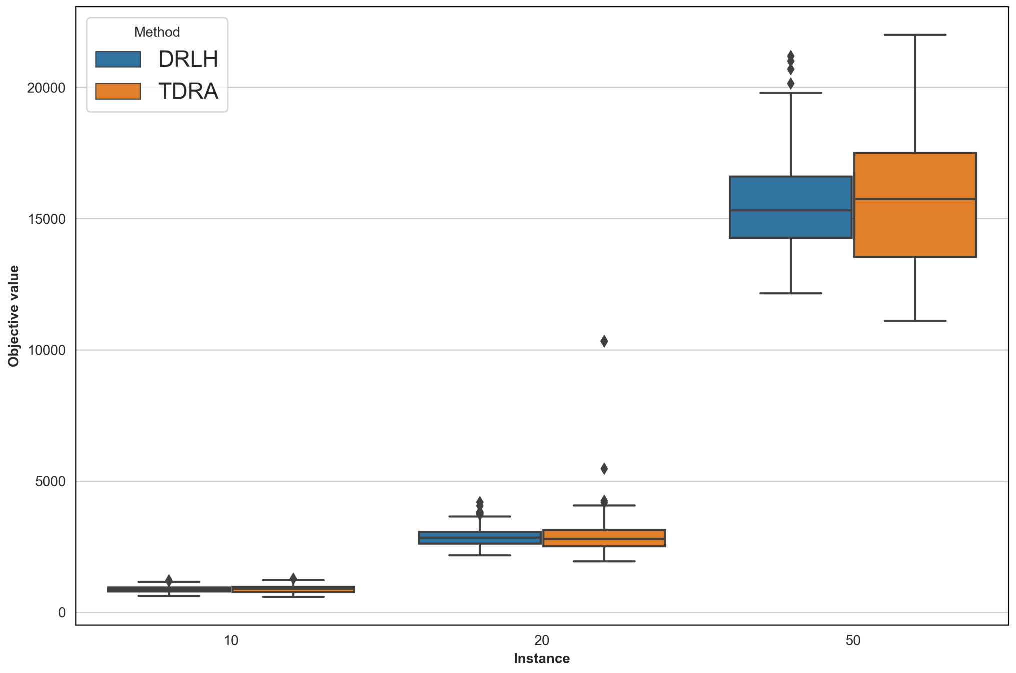


Figure 6.2: Boxplot showing a comparison of the distribution of objective values obtained by TDRA and DRLH on Set 3.

6.2 Results on the Test Set

Figure 6.3 shows the average improvement of DRLH over TDRA on the test set. All instances have been run 10 times. We see that TDRA performs better than DRLH on sizes 10 and 20. However, we see that DRLH significantly outperforms TDRA on size 50. Again, we see the same pattern, DRLH becomes progressively better as the problem size increases. Detailed numerical results on the test set are in Appendix B.2.3.

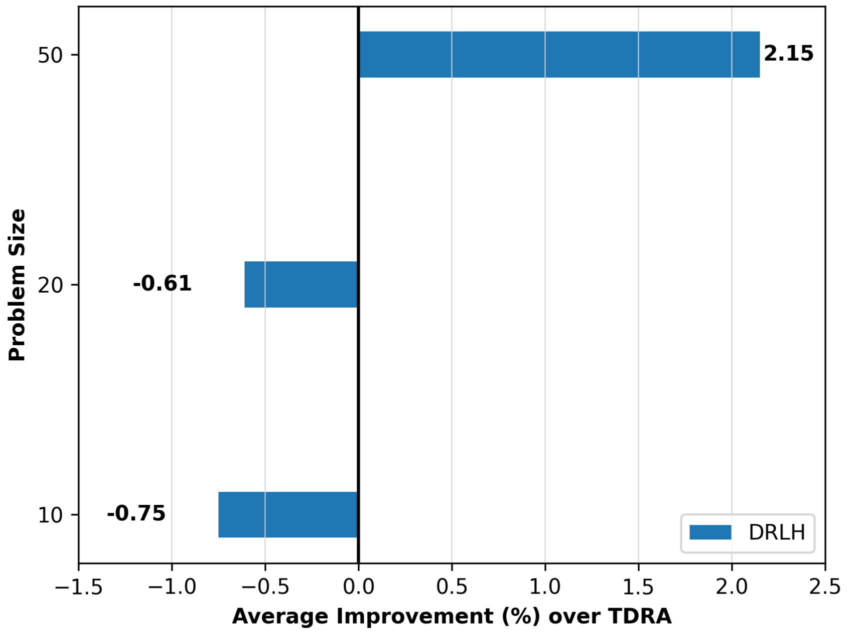


Figure 6.3: Results of DRLH on the test set using TDRA as baseline.

.

Figure 6.4 is a boxplot showing a comparison of the distribution of objective values obtained by TDRA and DRLH on the test set. All instances have been run 10 times. The central lines represent the median values, the box boundaries indicate the interquartile range (25th and 75th percentiles), and the whiskers extend to the minimum and maximum values, excluding outliers. We observe the same behavior for the test set and set 3. TDRA and DRLH perform similarly for size 10. For sizes 20 and 50 DRLH shows a more consistent performance, despite its lower best-case results. DRLH achieves a better median value for size 50.

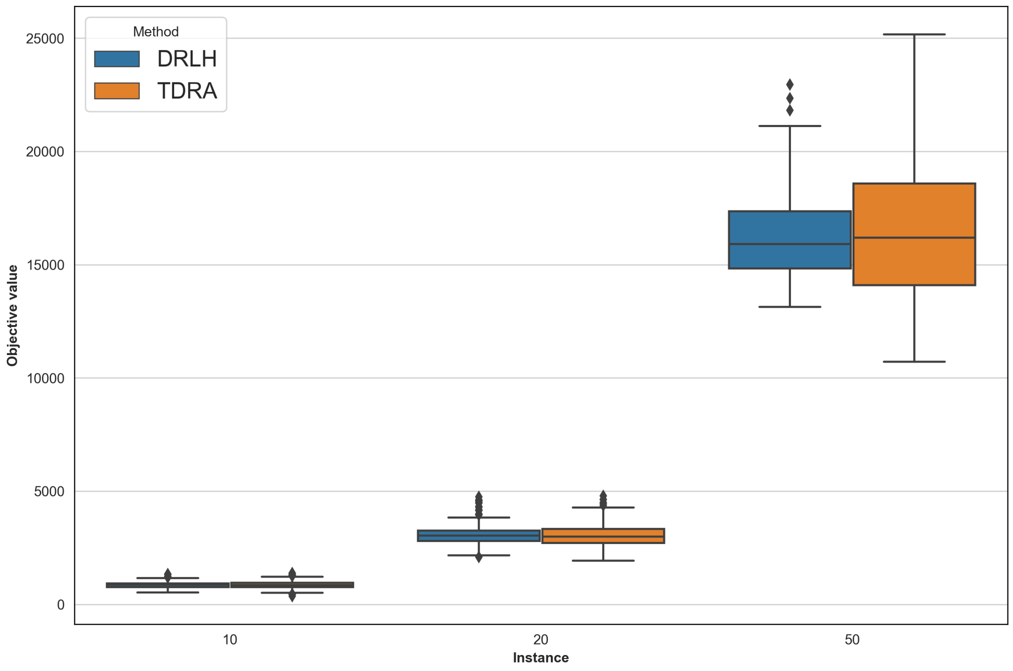
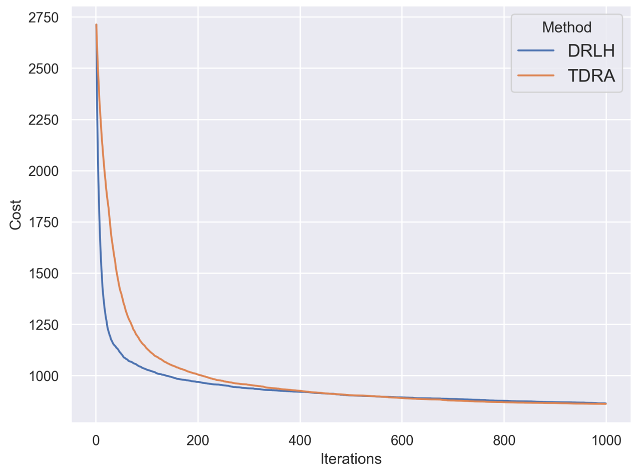


Figure 6.4: Boxplot showing a comparison of the distribution of objective values obtained by TDRA and DRLH on the test set.

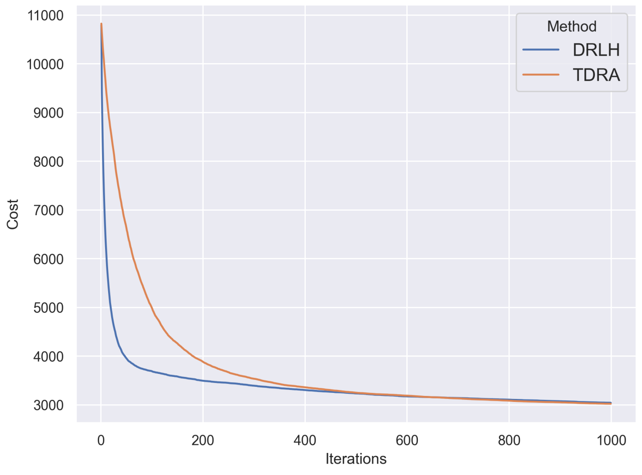
6.3 Performance Results

Figure 6.5 shows the average minimum costs of all instances in set 3 and the test set. All instances have been run 10 times. We observe that DRLH converges quicker than TDRA for all sizes. This trend seems to increase with the problem size. For sizes 10 and 20 DRLH starts stagnating after around 50 iterations. TDRA starts stagnating around iteration 100 for size 10 and around iteration 150 for size 20. For size 50, DRLH starts stagnating after about 100 iterations, while TDRA starts stagnating around 250 iterations.

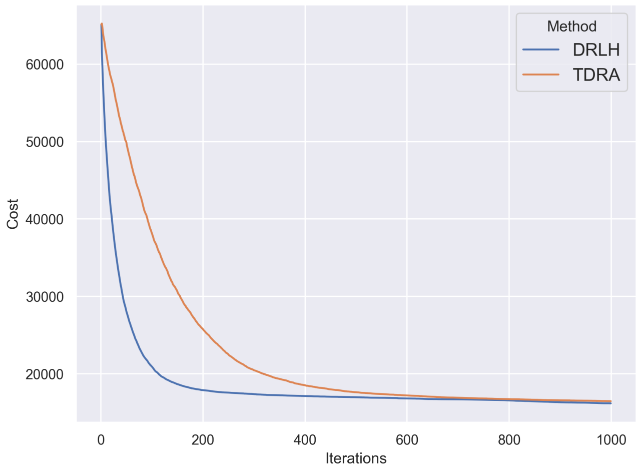
This trend is likely attributed to the micro-level heuristic selection employed by DRLH. While TDRA needs to explore the different heuristics to identify the best ones, DRLH leverages the current search state information and selects the most promising heuristics from the beginning. As a result, DRLH benefits from more effective heuristic choices early on, leading to quicker improvements in solution quality.



(a) 10 customer nodes, 1000 iterations



(b) 20 customer nodes, 1000 iterations



(c) 50 customer nodes, 1000 iterations

Figure 6.5: Average performance of TDRA and DRLH on both baseline sets.

6.4 TDRA's Performance on Small Problem Sizes

TDRA outperforms DRLH on the test set for sizes 10 and 20. There are probably two main causes for this performance.

Firstly, there is a small set of heuristics. Kallestad et al. (2023) showed that ALNS performed significantly worse when an extended set of heuristics were in play. This is probably because there are too many heuristics for ALNS to explore them all accurately and identify the best ones. TDRA also has this problem since it uses the same adaptive layer as ALNS. However, the heuristic set for the FTRPD is relatively small, meaning this problem might be mitigated.

Secondly, smaller problem sizes have smaller solution spaces than larger problems. This might mean that TDRA does not need micro-level heuristic selection to find good solutions for these problems. Macro-level heuristic selection might be sufficient for smaller problems, given that the same number of steps is used for problems of all sizes. Each heuristic will hold more value in larger problems where there is a vast search space. This is because more heuristics must be effective to find a good solution. There might be a higher need for micro-level heuristic selection when the solution space is large. This could also explain why DRLH becomes better as the problem size increases, as evidenced by Kallestad et al. (2023), Figure 6.1, and Figure 6.3.

Chapter 7

Conclusion and Future Work

In this thesis, we applied the Truck and Drone Routing Algorithm (TDRA) and Deep Reinforcement Learning Hyperheuristic (DRLH) to the Flexible Traveling Repairman Problem with Drones (FTRPD). The TDRA was implemented within a Python 3.10 environment in line with the methodology proposed by Moshref-Javadi et al. (2020), while the DRLH framework was adapted from Kallestad et al. (2023). Efforts were made to retain the original implementation of DRLH to assess its adaptability to a new problem. This led us to preserve the general state representation and most of the hyperparameters. Nonetheless, a few modifications were introduced, including an entropy coefficient to the total loss of the Proximal Policy Optimization (PPO) component and an adjustment to the learning rate. Additionally, due to the misalignment between our goal of cost minimization and the agent's default object of maximizing its return, we experimented with an alternative reward function. Details can be found in Appendix A.

Our experimental work involved performance comparisons using two baseline sets. In both instances, a clear pattern emerged with the effectiveness of DRLH increasing as the problem size increased. This trend reflects the findings of Kallestad et al. (2023), suggesting that DRLH could apply to larger real-world problems. DRLH was the more consistent model, and for all problem sizes, DRLH was able to converge in fewer iterations than TDRA in terms of average objective.

Future research should focus on larger instances of the FTRPD. It would be interesting to explore whether the performance of DRLH continues to improve over TDRA as the problem size increases. Kallestad et al. (2023) have trained models for sizes up to 500 for other vehicle routing problems and observed continuous improvements in DRLH over

Adaptive Large Neighborhood Search (ALNS). It could be a plausible assumption to expect a similar pattern for the FTRPD. Moreover, exploring the influence of various parameters – such as the number of drones, the drone speed, and the coverage percentage – on the performance of both TDRA and DRLH could be insightful. Lastly, it could be beneficial to train DRLH models using the Elliptical Customer Assignment (ECA) heuristic, which may help to find better objective values, particularly for larger problem sizes.

Glossary

incumbent The current solution in a search.

MinSum Objective function trying to minimize the waiting time of all customers..

non-improving iterations Iterations where the incumbent solution is not improved.

List of Acronyms and Abbreviations

ALNS Adaptive Large Neighborhood Search.

CVRP Capacitated Vehicle Routing Problem.

DRL Deep Reinforcement Learning.

DRLH Deep Reinforcement Learning Hyperheuristic.

ECA Elliptical Customer Assignment.

FTRPD Flexible Traveling Repairman Problem with Drones.

GA Genetic Algorithms.

LNS Large Neighborhood Search.

MLP MultiLayer perceptron.

PPO Proximal Policy Optimization.

RL Reinforcement Learning.

SA Simulated Annealing.

TDRA Truck and Drone Routing Algorithm.

TRP Traveling Repairman Problem.

TRPD Traveling Repairman Problem with Drones.

TSP Traveling Salesman Problem.

UAV Unmanned Aerial Vehicle.

VRPTWDR Vehicle Routing Problem with Time Windows and Delivery Robots.

Bibliography

Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. Understanding the impact of entropy on policy optimization, 2019.

Stephanie Alvarez, Angel Juan, Jésica Armas, Daniel Silva, and Daniel Riera. Metaheuristics in telecommunication systems: Network design, routing, and allocation problems. *IEEE Systems Journal*, PP:1–10, 01 2018. doi: 10.1109/JSYST.2017.2788053.

Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013. ISSN 1476-9360. doi: 10.1057/jors.2013.71.
URL: `https://doi.org/10.1057/jors.2013.71`.

Edmund K. Burke, Matthew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R. Woodward. *A Classification of Hyper-Heuristic Approaches: Revisited*, pages 453–477. Springer International Publishing, Cham, 2019.

Cheng Chen, Emrah Demir, and Yuan Huang. An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and delivery robots. *European Journal of Operational Research*, 294(3):1164–1180, 2021. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2021.02.027.
URL: `https://www.sciencedirect.com/science/article/pii/S037722172100120X`.

John H. Drake, Ahmed Kheiri, Ender Özcan, and Edmund K. Burke. Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, 285(2):405–428, 2020. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2019.07.073.
URL: `https://www.sciencedirect.com/science/article/pii/S0377221719306526`.

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

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, 2018.

Walter Gander, Gene H. Golub, and Rolf Strebel. Least-squares fitting of circles and ellipses. *BIT Numerical Mathematics*, 34(4):558–578, Dec 1994. ISSN 1572-9125. doi: 10.1007/BF01934268.
URL: `https://doi.org/10.1007/BF01934268`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart Russell, and Anca Dragan. Inverse reward design, 2020.

Kees Jacobs, Shannon Warner, Marc Rietra, Lindsey Mazza, Jerome Buvat, Amol Khadikar, Sumit Cherian, and Yashwardhan Khemka. The last-mile delivery challenge. *Capgemini Research Institute*, pages 1–40, 2019.

Jakob Kallestad, Ramin Hasibi, Ahmad Hemmati, and Kenneth Sörensen. A general deep reinforcement learning hyperheuristic framework for solving combinatorial optimization problems. *European Journal of Operational Research*, 309(1):446–468, 2023. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2023.01.017.
URL: `https://www.sciencedirect.com/science/article/pii/S037722172300036X`.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi: 10.1126/science.220.4598.671.
URL: `https://www.science.org/doi/abs/10.1126/science.220.4598.671`.

Lora Kolodny. Zipline raises $25 million to deliver medical supplies by drone, 2016.
URL: `https://techcrunch.com/2016/11/09/zipline-raises-25-million-to-deliver-medical-supplies-by-drone/`.

Frederic Lardinois. Amazon starts prime air drone delivery trial in the uk — but only with two beta users, 2016.
URL: `https://techcrunch.com/2016/12/14/amazons-prime-air-delivery-uk/`.

Alan Levine. Alphabet's project wing delivery drones to be tested in u.s., 2016.
URL: `https://www.bloomberg.com/news/articles/2016-08-02/google-s-project-wing-delivery-drones-to-be-tested-at-u-s-site?leadSource=uverify%20wall`.

Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *International Conference on Learning Representations*, 2020.

URL: `https://openreview.net/forum?id=BJe1334YDH`.

Mohammad Moshref-Javadi, Ahmad Hemmati, and Matthias Winkenbach. A truck and drones model for last-mile delivery: A mathematical model and heuristic approach. *Applied Mathematical Modelling*, 80:290–318, 2020. ISSN 0307-904X. doi: https://doi.org/10.1016/j.apm.2019.11.020.

URL: `https://www.sciencedirect.com/science/article/pii/S0307904X19306936`.

Michele Ottomanelli, Leonardo Caggiani, Mario Marinelli, and Mauro Dell'Orco. En-route truck-drone parcel delivery for optimal vehicle routing strategies. *IET Intelligent Transport Systems*, 12, 12 2017. doi: 10.1049/iet-its.2017.0227.

David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2005.09.012.

URL: `https://www.sciencedirect.com/science/article/pii/S0305054805003023`.

Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40:455–472, 11 2006. doi: 10.1287/trsc.1050.0135.

Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume B. Springer Science & Business Media, 1. edition, 2003.

Gerhard Schrimpf, Johannes Schneider, Hermann Stamm-Wilbrandt, and Gunter Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171, 2000. ISSN 0021-9991. doi: https://doi.org/10.1006/jcph.1999.6413.

URL: `https://www.sciencedirect.com/science/article/pii/S0021999199964136`.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

URL: `http://dblp.uni-trier.de/db/journals/corr/corr1707.html#SchulmanWDRK17`.

Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In Michael Maher and Jean-Francois Puget, editors, *Principles and Practice of Constraint Programming — CP98*, pages 417–431, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. ISBN 978-3-540-49481-2.

Alok Kumar Shukla, Diwakar Tripathi, B. Ramachandra Reddy, and D. Chandramohan. A study on metaheuristics approaches for gene selection in microarray data: algorithms, applications and open challenges. *Evolutionary Intelligence*, 13(3):309–329, Sep 2020. ISSN 1864-5917. doi: 10.1007/s12065-019-00306-6.
URL: `https://doi.org/10.1007/s12065-019-00306-6`.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

Kenneth Sörensen and Fred Glover. *Metaheuristics*, pages 960–970. Springer, 01 2013. ISBN 978-1-4419-1137-7. doi: 10.1007/978-1-4419-1153-7_1167.

Yuchang Zhang, Ruibin Bai, Rong Qu, Chaofan Tu, and Jiahuan Jin. A deep reinforcement learning based hyper-heuristic for combinatorial optimisation with uncertainties. *European Journal of Operational Research*, 300(2):418–427, 2022. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2021.10.032.
URL: `https://www.sciencedirect.com/science/article/pii/S0377221721008821`.

Appendix A

A Study of Different Reward Functions

A.1 R_t^{5310}

The reward function, R_t^{5310}, proposed by Kallestad et al. (2023) is designed to incentivize the agent by awarding five points for improving the global best solution, three points for enhancing the incumbent solution, a single point for discovering an unseen solution, and zero points otherwise.

All figures discussed herein have been smoothed to quickly identify data trends, with the instance number displayed on the y-axis.

Experiments were conducted by training a DRLH agent on 1000 instances of size 10. Figure A.1 demonstrates the gradual improvement in reward during the training process, with a significant surge between instances 200-400. As anticipated, the agent successfully learns to maximize its cumulative reward. However, this positive learning behavior exhibits unintended consequences, as seen in Figure A.2, which depicts the cost associated with the optimal global solution for each instance. Here, a considerable cost increase is observed between instances 200-400. This contradicts our goal of minimizing this cost, revealing a significant discord between our aim and the agent´s objective.

This disconnect could be attributed to various factors. The first of these is the small size of the problem, size 10. The room for improvement is limited as the initial cost is relatively close to the optimal global solution. As such, it becomes advantageous for the agent to incrementally elevate the objective at each step rather than make a significant leap. Evidence of this behavior can be seen in Figure A.5, where the number of improvements skyrockets between instances 200-400. Within this timeframe, both the

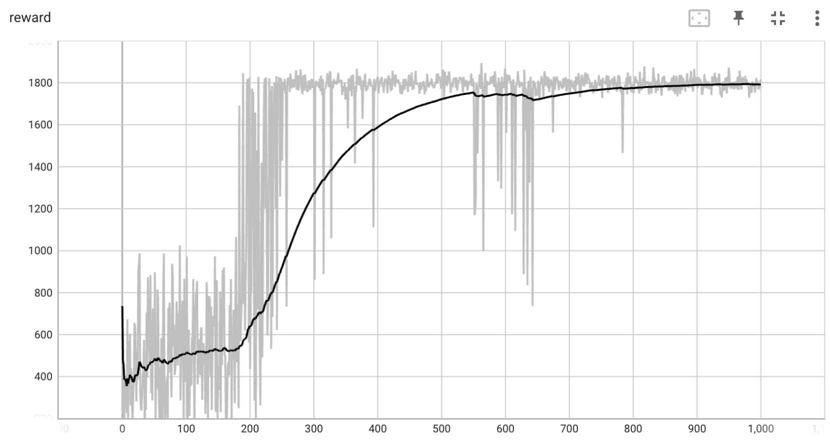


Figure A.1: DRLH agent's return on training set of 1000 instances.

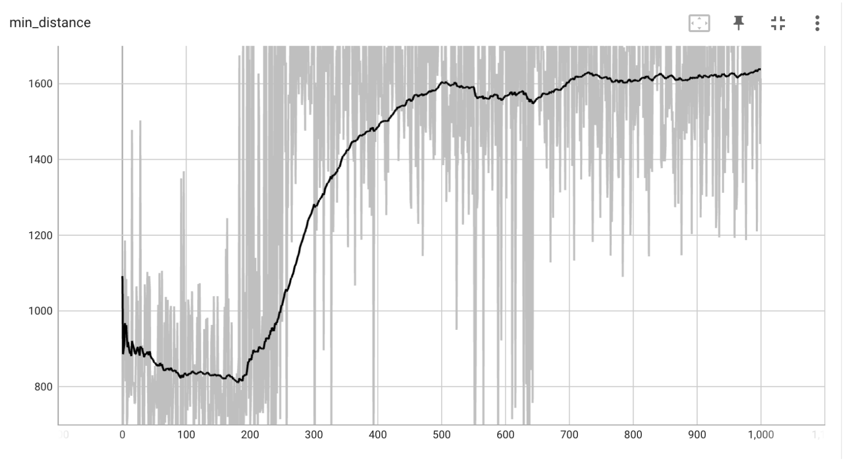


Figure A.2: DRLH agent's minimum distance on training set of 1000 instances.

return and the minimum distance experience a substantial increase. Furthermore, Figure A.3 shows a dramatic decline in the number of improvements on the global best solution within the same period.

From Figure A.4, we can see that the number of seen solutions increases dramatically in the timeframe 200-400. The agent learns that finding new solutions at almost every step is more beneficial for its return. This means the agent will focus much more on exploration than exploitation. As mentioned earlier, it is essential to balance diversification and intensification to find high-quality solutions in metaheuristics. Focusing too much on exploration will result in seeing many solutions, but the solutions will be of low quality. This is one of the reasons the best-found objective increases in the same timeframe as the number of seen solutions increases.

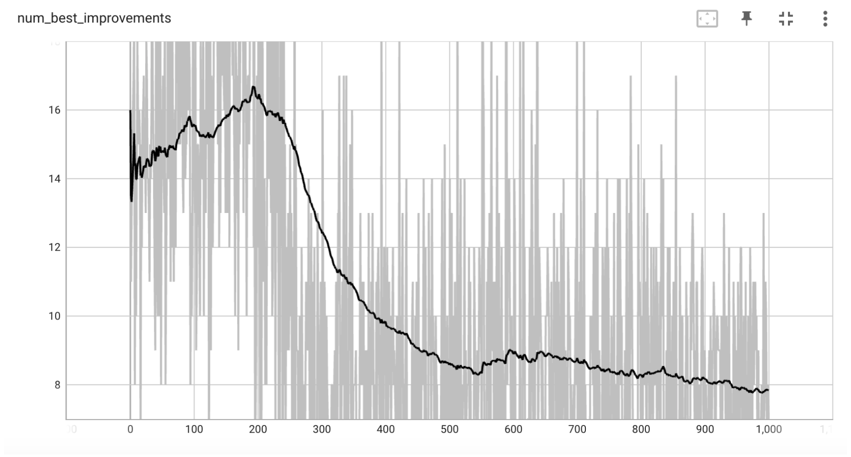


Figure A.3: DRLH agent's number of improvements on the global best solution on training set of 1000 instances.

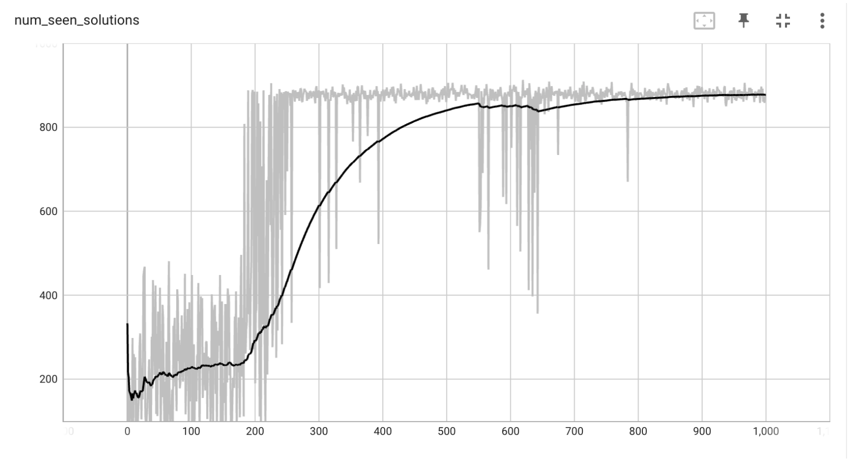


Figure A.4: DRLH agent's number of seen solutions on training set of 1000 instances.

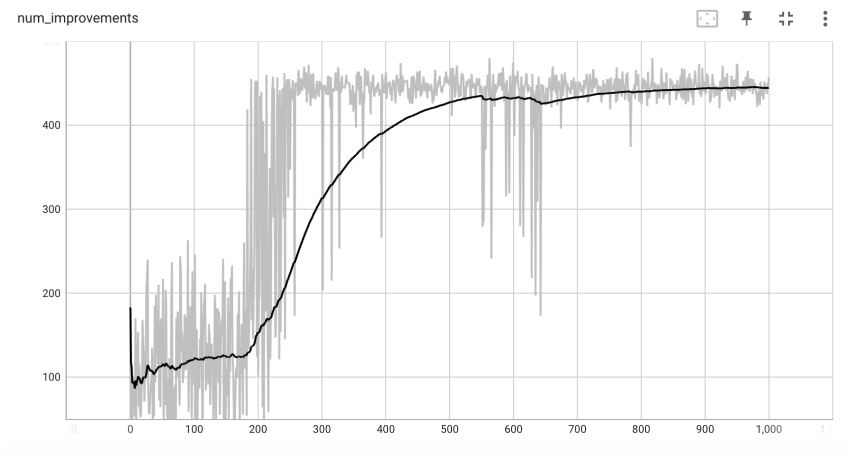


Figure A.5: DRLH agent´s number of improvements on the incumbent solution during training on 1000 instances.

A.2 R_t^{PM}

As proposed by Kallestad et al. (2023), the R_t^{PM} reward function rewards one point when the incumbent solution is improved and a penalty of minus one point when it's not. This function leans towards intensification, but thanks to the structure of the PPO framework that relies on future discounted rewards rather than only immediate rewards, R_t^{PM} can also select heuristics for diversification. The agent may choose a heuristic with a low immediate reward if it yields a high return in the long run. This function addresses the issue of an excessive number of seen solutions, a problematic aspect of the R_t^{5310} reward function.

Nonetheless, the R_t^{PM} reward function has its shortcomings. Similar to the R_t^{5310} function, it learns to improve the incumbent solution slightly at every step to maximize its return, which does not align with the goal of minimizing the objective function. Consequently, as the return and number of improvements increase significantly, the minimum distance also rises. Again, we face a misalignment between our objective of minimizing the objective and the agent's aim to maximize its return.

A.3 R_t^{210}

R_t^{210} is the reward function proposed by Moshref-Javadi et al. (2020) for use in the TDRA. It rewards two points when the global best solution is improved, one point when the incumbent solution is improved, and zero points otherwise. Similarly to R_t^{PM}, it is a reward function that favors intensification and addresses the issue of an excessive number of seen solutions. This reward function is beneficial because it is used in TDRA. This would have enabled a fair comparison because you could not argue that one of the algorithms uses a better reward function.

Nevertheless, the R_t^{210} reward function has limitations, sharing similar issues with R_t^{PM} and R_t^{5310}. In all these cases, the agent identifies that incremental progress toward a better objective yields higher returns. A consistent pattern emerges, marked by a notable upswing in the number of improvements, the returns, and the minimum distance within the same timeframe.

Appendix B

Detailed Results on Baseline Sets

B.1 Detailed Results on Set 3

B.1.1 Size 10

Table B.1 gives the detailed numerical results of TDRA on Set 3 instances of size 10. Column *Average Solution* reports the average of the best-found objective values for 10 runs on each instance.

Instance	c	Model	TRP	Average Cost	Best Cost	Run Time (sec)	Customers served by UAVs	Δ(%)
TRP-S10-R1	10	FTRPD	1850	1036.2	918	13.3	6	50.38
TRP-S10-R2	10	FTRPD	2937	915.2	836	11.5	4	71.54
TRP-S10-R3	10	FTRPD	3019	955.6	827	11.0	6	72.61
TRP-S10-R4	10	FTRPD	3116	870.7	651	11.7	5	79.11
TRP-S10-R5	10	FTRPD	2561	795.2	628	11.9	5	75.48
TRP-S10-R6	10	FTRPD	3452	1003.8	779	11.2	5	77.43
TRP-S10-R7	10	FTRPD	2074	888.5	832	10.5	3	59.88
TRP-S10-R8	10	FTRPD	2780	910.5	791	7.7	3	71.55
TRP-S10-R9	10	FTRPD	3354	973.9	807	7.8	3	75.94
TRP-S10-R10	10	FTRPD	3263	850.3	628	9.8	4	80.75
TRP-S10-R11	10	FTRPD	3321	904.7	675	10.4	5	79.67
TRP-S10-R12	10	FTRPD	3066	902.2	753	12.0	5	75.44
TRP-S10-R13	10	FTRPD	3123	1059.6	904	11.2	4	71.05
TRP-S10-R14	10	FTRPD	2020	677.2	585	10.5	4	71.04
TRP-S10-R15	10	FTRPD	2240	821.7	708	11.7	5	68.39
TRP-S10-R16	10	FTRPD	2461	857.4	709	9.2	4	71.19
TRP-S10-R17	10	FTRPD	2728	812.9	666	9.6	4	75.59
TRP-S10-R18	10	FTRPD	2407	716.7	672	9.7	5	72.08
TRP-S10-R19	10	FTRPD	3232	925.3	813	11.0	6	74.85
TRP-S10-R20	10	FTRPD	2075	674.7	621	12.5	5	70.07

Table B.1: Set 3 size 10 results for TDRA with baseline parameters of drone speed 1.5 and coverage 25%.

Table B.2 gives the detailed numerical results of DRLH on Set 3 instances of size 10. Column *Average Solution* reports the average of the best-found objective values for 10 runs on each instance.

Instance	c	Model	TRP	Average Cost	Best Cost	Run Time (sec)	Customers served by UAVs	Δ(%)
TRP-S10-R1	10	FTRPD	1850	999.0	915	12.3	6	50.54
TRP-S10-R2	10	FTRPD	2937	883.2	709	12.4	4	75.86
TRP-S10-R3	10	FTRPD	3019	954.8	831	12.2	6	72.47
TRP-S10-R4	10	FTRPD	3116	865.7	776	12.8	5	75.1
TRP-S10-R5	10	FTRPD	2561	827.9	623	12.7	5	75.67
TRP-S10-R6	10	FTRPD	3452	981.0	831	12.2	5	75.93
TRP-S10-R7	10	FTRPD	2074	965.9	858	12.0	3	58.63
TRP-S10-R8	10	FTRPD	2780	900.6	775	12.4	3	72.12
TRP-S10-R9	10	FTRPD	3354	926.0	751	12.5	3	77.61
TRP-S10-R10	10	FTRPD	3263	895.6	772	12.2	4	76.34
TRP-S10-R11	10	FTRPD	3321	836.2	697	12.8	5	79.01
TRP-S10-R12	10	FTRPD	3066	847.2	753	12.4	5	75.44
TRP-S10-R13	10	FTRPD	3123	1013.8	904	12.5	4	71.05
TRP-S10-R14	10	FTRPD	2020	758.8	636	13.5	4	68.51
TRP-S10-R15	10	FTRPD	2240	770.2	750	13.2	5	66.52
TRP-S10-R16	10	FTRPD	2461	875.1	736	12.3	4	70.09
TRP-S10-R17	10	FTRPD	2728	782.1	680	13.2	4	75.07
TRP-S10-R18	10	FTRPD	2407	731.2	657	12.5	5	72.7
TRP-S10-R19	10	FTRPD	3232	897.5	807	11.9	6	75.03
TRP-S10-R20	10	FTRPD	2075	807.9	646	12.9	5	68.87

Table B.2: Set 3 size 10 results for DRLH with baseline parameters of drone speed 1.5 and coverage 25%.

B.1.2 Size 20

Table B.3 gives the detailed numerical results of TDRA on Set 3 instances of size 20. Column *Average Solution* reports the average of the best-found objective values for 10 runs on each instance.

Instance	c	Model	TRP	Average Cost	Best Cost	Run Time (sec)	Customers served by UAVs	Δ(%)
TRP-S20-R1	20	FTRPD	10041	3013.7	2452	34.6	7	75.58
TRP-S20-R2	20	FTRPD	11879	2574.0	2390	36.9	5	79.88
TRP-S20-R3	20	FTRPD	13070	3150.4	2437	31.6	5	81.35
TRP-S20-R4	20	FTRPD	9189	2597.4	1935	38.8	9	78.94
TRP-S20-R5	20	FTRPD	9195	2702.0	2449	32.7	5	73.37
TRP-S20-R6	20	FTRPD	12026	2765.7	2478	29.4	3	79.39
TRP-S20-R7	20	FTRPD	10328	2366.3	2109	31.3	6	79.58
TRP-S20-R8	20	FTRPD	11894	3117.1	2640	38.7	11	77.8
TRP-S20-R9	20	FTRPD	11848	2843.4	2082	32.4	6	82.43
TRP-S20-R10	20	FTRPD	11682	2743.4	2263	38.0	6	80.63
TRP-S20-R11	20	FTRPD	10308	2560.8	2194	31.7	5	78.72
TRP-S20-R12	20	FTRPD	12313	3346.2	2867	39.2	7	76.72
TRP-S20-R13	20	FTRPD	10898	2901.1	2582	35.7	5	76.31
TRP-S20-R14	20	FTRPD	11305	2937.0	2511	28.7	3	77.79
TRP-S20-R15	20	FTRPD	11436	2600.3	2334	38.7	8	79.59
TRP-S20-R16	20	FTRPD	12579	2897.9	2144	36.8	5	82.96
TRP-S20-R17	20	FTRPD	11383	2577.0	2222	35.0	3	80.48
TRP-S20-R18	20	FTRPD	10671	2881.3	2420	25.5	3	77.32
TRP-S20-R19	20	FTRPD	8354	3309.0	2763	37.5	9	66.93
TRP-S20-R20	20	FTRPD	13424	2516.5	2042	32.0	3	84.79

Table B.3: Set 3 size 20 results for TDRA with baseline parameters of drone speed 1.5 and coverage 25%.

Table B.4 gives the detailed numerical results of DRLH on Set 3 instances of size 20. Column *Average Solution* reports the average of the best-found objective values for 10 runs on each instance.

Instance	c	Model	TRP	Average Cost	Best Cost	Run Time (sec)	Customers served by UAVs	Δ(%)
TRP-S20-R1	20	FTRPD	10041	2720.3	2408	33.1	8	76.02
TRP-S20-R2	20	FTRPD	11879	2581.0	2227	55.6	10	81.25
TRP-S20-R3	20	FTRPD	13070	2880.4	2497	41.2	5	80.9
TRP-S20-R4	20	FTRPD	9189	2632.8	2168	39.3	7	76.41
TRP-S20-R5	20	FTRPD	9195	2675.4	2468	51.4	8	73.16
TRP-S20-R6	20	FTRPD	12026	2702.0	2429	58.2	5	79.8
TRP-S20-R7	20	FTRPD	10328	2513.3	2296	84.5	4	77.77
TRP-S20-R8	20	FTRPD	11894	2657.9	2250	51.1	6	81.08
TRP-S20-R9	20	FTRPD	11848	3089.0	2760	39.0	3	76.7
TRP-S20-R10	20	FTRPD	11682	2797.3	2421	52.9	8	79.28
TRP-S20-R11	20	FTRPD	10308	2742.5	2275	57.0	6	77.93
TRP-S20-R12	20	FTRPD	12313	3552.0	3002	38.7	2	75.62
TRP-S20-R13	20	FTRPD	10898	3031.6	2301	35.1	4	78.89
TRP-S20-R14	20	FTRPD	11305	2864.0	2590	36.7	3	77.09
TRP-S20-R15	20	FTRPD	11436	2931.0	2731	50.5	3	76.12
TRP-S20-R16	20	FTRPD	12579	2762.2	2427	59.2	5	80.71
TRP-S20-R17	20	FTRPD	10324	2547.0	2432	46.1	2	76.44
TRP-S20-R18	20	FTRPD	10671	2724.6	2462	31.4	3	76.93
TRP-S20-R19	20	FTRPD	8354	3149.4	2886	36.2	3	65.45
TRP-S20-R20	20	FTRPD	13424	2712.4	2332	42.3	4	82.63

Table B.4: Set 3 size 20 results for DRLH with baseline parameters of drone speed 1.5 and coverage 25%.

B.1.3 Size 50

Table B.5 gives the detailed numerical results of TDRA on Set 3 instances of size 50. Column *Average Solution* reports the average of the best-found objective values for 10 runs on each instance.

Instance	c	Model	TRP	Average Cost	Best Cost	Run Time (sec)	Customers served by UAVs	Δ(%)
TRP-S50-R1	50	FTRPD	65755	16047.2	12619	269.5	2	80.81
TRP-S50-R2	50	FTRPD	65060	13861.2	12146	257.1	3	81.33
TRP-S50-R3	50	FTRPD	64643	14763.8	13248	233.0	2	79.51
TRP-S50-R4	50	FTRPD	73502	15911.0	14269	290.3	2	80.59
TRP-S50-R5	50	FTRPD	70964	17057.8	14609	436.3	6	79.41
TRP-S50-R6	50	FTRPD	65328	15405.4	13258	227.1	2	79.71
TRP-S50-R7	50	FTRPD	73339	13338.4	11803	133.5	2	83.91
TRP-S50-R8	50	FTRPD	60746	15322.0	12525	348.5	4	79.38
TRP-S50-R9	50	FTRPD	68202	17847.0	12596	411.9	3	81.53
TRP-S50-R10	50	FTRPD	66125	15934.2	13154	363.7	9	80.11
TRP-S50-R11	50	FTRPD	66880	14537.8	11009	232.4	5	83.54
TRP-S50-R12	50	FTRPD	60286	16473.4	14594	435.9	10	75.79
TRP-S50-R13	50	FTRPD	73468	14663.0	12029	219.2	2	83.63
TRP-S50-R14	50	FTRPD	60412	15155.6	12880	238.4	5	78.68
TRP-S50-R15	50	FTRPD	73136	16770.8	13814	358.5	2	81.11
TRP-S50-R16	50	FTRPD	63651	17167.6	14431	295.0	8	77.33
TRP-S50-R17	50	FTRPD	65751	13051.6	11910	213.3	3	81.89
TRP-S50-R18	50	FTRPD	64581	18726.6	16363	302.7	4	74.66
TRP-S50-R19	50	FTRPD	65089	14686.2	12731	205.9	2	80.44
TRP-S50-R20	50	FTRPD	72786	15124.6	12660	288.5	6	82.61

Table B.5: Set 3 size 50 results for TDRA with baseline parameters of drone speed 1.5 and coverage 25%.

Table B.6 gives the detailed numerical results of DRLH on Set 3 instances of size 50. Column *Average Solution* reports the average of the best-found objective values for 10 runs on each instance.

Instance	c	Model	TRP	Average Cost	Best Cost	Run Time (sec)	Customers served by UAVs	Δ(%)
TRP-S50-R1	50	FTRPD	65755	14825.5	13974	339.5	3	78.75
TRP-S50-R2	50	FTRPD	65060	16842.8	13589	304.7	3	79.11
TRP-S50-R3	50	FTRPD	64643	14147.0	13500	350.5	2	79.12
TRP-S50-R4	50	FTRPD	73502	16142.0	15305	344.5	6	79.18
TRP-S50-R5	50	FTRPD	70964	16716.3	15490	280.7	4	78.17
TRP-S50-R6	50	FTRPD	65328	16298.0	14024	211.4	2	78.53
TRP-S50-R7	50	FTRPD	73339	13698.3	12898	293.5	2	82.41
TRP-S50-R8	50	FTRPD	60746	14916.7	14233	338.1	3	76.57
TRP-S50-R9	50	FTRPD	68202	17644.2	13614	379.7	2	80.04
TRP-S50-R10	50	FTRPD	66125	17075.0	14676	315.9	2	77.81
TRP-S50-R11	50	FTRPD	66880	15609.8	14778	375.3	3	77.90
TRP-S50-R12	50	FTRPD	60286	14625.2	12150	449.6	2	79.85
TRP-S50-R13	50	FTRPD	73468	14173.5	13314	390.6	2	81.88
TRP-S50-R14	50	FTRPD	60412	15448.3	14754	335.2	5	75.58
TRP-S50-R15	50	FTRPD	73136	16304.3	14490	353.0	3	81.11
TRP-S50-R16	50	FTRPD	63651	15860.8	14726	352.6	4	77.33
TRP-S50-R17	50	FTRPD	65751	13669.6	13266	231.4	2	79.82
TRP-S50-R18	50	FTRPD	64581	17191.4	16783	332.4	7	74.01
TRP-S50-R19	50	FTRPD	65089	14441.2	14089	306.1	8	78.35
TRP-S50-R20	50	FTRPD	72786	14356.0	13680	341.8	2	81.21

Table B.6: Set 3 size 50 results for DRLH with baseline parameters of drone speed 1.5 and coverage 25%.

B.2 Detailed Results on the Test Set

B.2.1 Size 10

Table B.7 gives the detailed numerical results of TDRA on the test set instances of size 10. Column *Average Solution* reports the average of the best-found objective values for 10 runs on each instance.

Table B.8 gives the detailed numerical results of DRLH on the test set instances of size 10. Column *Average Solution* reports the average of the best-found objective values for 10 runs on each instance.

B.2.2 Size 20

Table B.9 gives the detailed numerical results of TDRA on the test set instances of size 20. Column *Average Solution* reports the average of the best-found objective values for 10 runs on each instance.

Table B.10 gives the detailed numerical results of DRLH on the test set instances of size 20. Column *Average Solution* reports the average of the best-found objective values for 10 runs on each instance.

Instance	c	Model	TRP	Average Cost	Best Cost	Run Time (sec)	Customers served by UAVs	Δ(%)
Test-S10-R1	10	FTRPD	3043	1174.8	929	9.8	6	69.47
Test-S10-R2	10	FTRPD	2653	811.0	715	9.6	4	73.05
Test-S10-R3	10	FTRPD	4002	906.2	751	10.9	4	81.23
Test-S10-R4	10	FTRPD	1994	594.1	459	8.1	4	76.98
Test-S10-R5	10	FTRPD	3187	743.0	642	8.9	4	79.86
Test-S10-R6	10	FTRPD	2456	754.9	638	11.0	5	74.02
Test-S10-R7	10	FTRPD	2331	840.6	764	9.1	5	67.22
Test-S10-R8	10	FTRPD	1849	703.5	592	11.0	5	67.98
Test-S10-R9	10	FTRPD	2055	937.9	906	10.9	4	55.91
Test-S10-R10	10	FTRPD	2604	791.6	657	9.5	6	74.77
Test-S10-R11	10	FTRPD	3031	873.8	765	10.2	5	74.76
Test-S10-R12	10	FTRPD	2937	775.3	626	10.4	5	78.69
Test-S10-R13	10	FTRPD	2563	874.5	756	11.8	6	70.5
Test-S10-R14	10	FTRPD	3579	762.4	673	9.5	4	81.2
Test-S10-R15	10	FTRPD	3054	973.7	860	10.3	5	71.84
Test-S10-R16	10	FTRPD	3498	987.4	789	10.7	7	77.44
Test-S10-R17	10	FTRPD	3842	850.1	778	10.3	5	79.75
Test-S10-R18	10	FTRPD	2417	731.9	702	9.3	5	70.96
Test-S10-R19	10	FTRPD	2855	981.9	908	11.0	6	68.2
Test-S10-R20	10	FTRPD	2801	799.7	711	10.0	5	74.62
Test-S10-R21	10	FTRPD	2364	731.4	635	13.5	6	73.14
Test-S10-R22	10	FTRPD	3056	987.3	769	10.9	4	74.84
Test-S10-R23	10	FTRPD	2412	1113.0	1011	11.1	5	58.08
Test-S10-R24	10	FTRPD	3969	919.0	835	11.5	7	78.96
Test-S10-R25	10	FTRPD	2972	831.7	680	9.8	5	77.12
Test-S10-R26	10	FTRPD	2774	901.6	850	10.4	4	69.36
Test-S10-R27	10	FTRPD	2284	738.9	649	9.3	5	71.58
Test-S10-R28	10	FTRPD	2504	743.7	628	8.6	6	74.92
Test-S10-R29	10	FTRPD	3219	974.6	889	10.5	6	72.38
Test-S10-R30	10	FTRPD	2116	866.2	703	10.1	6	66.78
Test-S10-R31	10	FTRPD	2343	868.9	715	12.1	5	69.48
Test-S10-R32	10	FTRPD	2807	874.7	773	8.9	4	72.46
Test-S10-R33	10	FTRPD	3617	928.2	796	8.6	4	77.99
Test-S10-R34	10	FTRPD	2958	1051.5	969	11.6	5	67.24
Test-S10-R35	10	FTRPD	2427	922.1	807	11.3	6	66.75
Test-S10-R36	10	FTRPD	2348	617.1	444	8.4	4	81.09
Test-S10-R37	10	FTRPD	2582	700.4	647	8.2	3	74.94
Test-S10-R38	10	FTRPD	2783	984.8	892	10.5	5	67.95
Test-S10-R39	10	FTRPD	2322	744.0	650	8.8	4	72.01
Test-S10-R40	10	FTRPD	2167	772.0	684	9.0	5	68.44
Test-S10-R41	10	FTRPD	3231	877.0	727	11.2	5	77.5
Test-S10-R42	10	FTRPD	3876	955.8	838	10.3	6	78.38
Test-S10-R43	10	FTRPD	2383	858.3	828	9.5	5	65.25
Test-S10-R44	10	FTRPD	3336	1066.2	934	10.3	3	72.0
Test-S10-R45	10	FTRPD	2388	615.1	570	9.6	5	76.13
Test-S10-R46	10	FTRPD	2833	553.6	364	8.5	6	87.15
Test-S10-R47	10	FTRPD	3271	1013.8	814	9.6	5	75.11
Test-S10-R48	10	FTRPD	2276	820.3	745	10.7	5	67.27
Test-S10-R49	10	FTRPD	2975	967.3	895	9.8	5	69.92
Test-S10-R50	10	FTRPD	3120	926.1	771	9.0	4	75.29

Table B.7: Test set size 10 results for TDRA with baseline parameters of drone speed 1.5 and coverage 25%.

Instance	c	Model	TRP	Average Cost	Best Cost	Run Time (sec)	Customers served by UAVs	Δ(%)
Test-S10-R1	10	FTRPD	3043	970.0	932	11.3	5	69.37
Test-S10-R2	10	FTRPD	2653	764.1	688	11.2	4	74.07
Test-S10-R3	10	FTRPD	4002	919.4	772	11.3	6	80.71
Test-S10-R4	10	FTRPD	1994	713.9	559	12.0	4	71.97
Test-S10-R5	10	FTRPD	3187	826.0	669	11.4	3	79.01
Test-S10-R6	10	FTRPD	2456	775.5	756	11.9	5	69.22
Test-S10-R7	10	FTRPD	2331	787.1	656	11.8	5	71.86
Test-S10-R8	10	FTRPD	1849	686.9	632	12.1	4	65.82
Test-S10-R9	10	FTRPD	2055	907.3	906	11.6	4	55.91
Test-S10-R10	10	FTRPD	2604	845.7	709	11.7	5	72.77
Test-S10-R11	10	FTRPD	3031	866.9	765	11.5	5	74.76
Test-S10-R12	10	FTRPD	2937	814.1	745	12.1	5	74.63
Test-S10-R13	10	FTRPD	2563	880.2	806	11.8	6	68.55
Test-S10-R14	10	FTRPD	3579	854.4	734	11.6	4	79.49
Test-S10-R15	10	FTRPD	3054	978.8	889	11.4	4	70.89
Test-S10-R16	10	FTRPD	3498	971.0	923	11.2	5	73.61
Test-S10-R17	10	FTRPD	3842	852.8	790	12.2	5	79.44
Test-S10-R18	10	FTRPD	2417	771.1	725	11.7	4	70.0
Test-S10-R19	10	FTRPD	2855	999.5	908	11.3	6	68.2
Test-S10-R20	10	FTRPD	2801	810.4	735	12.5	5	73.76
Test-S10-R21	10	FTRPD	2364	674.1	613	17.6	7	74.07
Test-S10-R22	10	FTRPD	3056	996.7	859	18.5	3	71.89
Test-S10-R23	10	FTRPD	2412	1177.1	1025	20.8	4	57.5
Test-S10-R24	10	FTRPD	3969	1211.0	850	20.3	6	78.58
Test-S10-R25	10	FTRPD	2972	799.8	605	21.6	5	79.64
Test-S10-R26	10	FTRPD	2774	852.8	814	18.9	6	70.66
Test-S10-R27	10	FTRPD	2284	769.7	649	18.7	5	71.58
Test-S10-R28	10	FTRPD	2504	781.1	694	18.6	4	72.28
Test-S10-R29	10	FTRPD	3219	933.9	835	24.4	7	74.06
Test-S10-R30	10	FTRPD	2116	875.4	683	21.0	6	67.72
Test-S10-R31	10	FTRPD	2343	834.9	699	16.7	5	70.17
Test-S10-R32	10	FTRPD	2807	819.9	757	21.2	5	73.03
Test-S10-R33	10	FTRPD	3617	921.6	728	22.8	4	79.87
Test-S10-R34	10	FTRPD	2958	960.1	925	14.3	5	68.73
Test-S10-R35	10	FTRPD	2427	879.9	769	12.8	6	68.31
Test-S10-R36	10	FTRPD	2348	674.2	529	11.6	5	77.47
Test-S10-R37	10	FTRPD	2582	734.8	692	12.1	4	73.2
Test-S10-R38	10	FTRPD	2783	922.3	881	13.1	5	68.34
Test-S10-R39	10	FTRPD	2322	810.5	734	12.7	5	68.39
Test-S10-R40	10	FTRPD	2167	786.1	679	12.2	4	68.67
Test-S10-R41	10	FTRPD	3231	856.1	786	13.7	6	75.67
Test-S10-R42	10	FTRPD	3876	989.5	889	15.0	5	77.06
Test-S10-R43	10	FTRPD	2383	852.8	779	14.9	7	67.31
Test-S10-R44	10	FTRPD	3336	1033.2	892	15.6	4	73.26
Test-S10-R45	10	FTRPD	2388	631.8	530	14.4	7	77.81
Test-S10-R46	10	FTRPD	2833	700.4	559	15.3	4	80.27
Test-S10-R47	10	FTRPD	3271	995.6	867	16.0	3	73.49
Test-S10-R48	10	FTRPD	2276	836.5	726	15.4	6	68.1
Test-S10-R49	10	FTRPD	2975	927.8	860	15.7	3	71.09
Test-S10-R50	10	FTRPD	3120	876.0	782	13.3	5	74.94

Table B.8: Test set size 10 results for DRLH with baseline parameters of drone speed 1.5 and coverage 25%.

Instance	c	Model	TRP	Average Cost	Best Cost	Run Time (sec)	Customers served by UAVs	Δ(%)
Test-S20-R1	20	FTRPD	11181	2976.1	2291	29.5	7	79.51
Test-S20-R2	20	FTRPD	12154	3169.2	2485	37.5	6	79.55
Test-S20-R3	20	FTRPD	10483	3028.3	2526	32.8	3	75.9
Test-S20-R4	20	FTRPD	10138	2848.7	2251	31.0	3	77.8
Test-S20-R5	20	FTRPD	10473	2830.7	2399	38.3	6	77.09
Test-S20-R6	20	FTRPD	10645	3138.8	2670	34.2	5	74.92
Test-S20-R7	20	FTRPD	15052	3196.2	2349	39.2	9	84.39
Test-S20-R8	20	FTRPD	11321	3126.5	2413	29.6	3	78.69
Test-S20-R9	20	FTRPD	12434	3059.9	2321	33.9	8	81.33
Test-S20-R10	20	FTRPD	10236	2991.4	2570	31.6	2	74.89
Test-S20-R11	20	FTRPD	9709	2688.6	2361	27.7	4	75.68
Test-S20-R12	20	FTRPD	11976	3025.9	2544	32.5	6	78.76
Test-S20-R13	20	FTRPD	10308	2858.4	2137	27.9	4	79.27
Test-S20-R14	20	FTRPD	8959	2806.8	2090	28.9	8	76.67
Test-S20-R15	20	FTRPD	11224	2410.9	2086	27.4	4	81.41
Test-S20-R16	20	FTRPD	12373	3375.0	2713	36.8	11	78.07
Test-S20-R17	20	FTRPD	10770	3278.1	2744	33.0	3	74.52
Test-S20-R18	20	FTRPD	9404	3113.0	2758	35.8	5	70.67
Test-S20-R19	20	FTRPD	9490	2811.7	2342	34.8	4	75.32
Test-S20-R20	20	FTRPD	13166	3873.3	2994	37.1	9	77.26
Test-S20-R21	20	FTRPD	11909	2610.2	1929	27.5	5	83.8
Test-S20-R22	20	FTRPD	10496	2859.0	2127	31.3	7	79.74
Test-S20-R23	20	FTRPD	12999	2896.0	2477	31.4	3	80.94
Test-S20-R24	20	FTRPD	10257	3035.2	2356	27.8	6	77.03
Test-S20-R25	20	FTRPD	9707	2574.5	2146	31.1	6	77.89
Test-S20-R26	20	FTRPD	10933	3467.0	2494	36.4	6	77.19
Test-S20-R27	20	FTRPD	11823	3401.6	2785	35.3	5	76.44
Test-S20-R28	20	FTRPD	9378	2991.8	2692	34.1	4	71.29
Test-S20-R29	20	FTRPD	9711	3457.8	2920	38.3	11	69.93
Test-S20-R30	20	FTRPD	11069	3228.7	2648	36.8	8	76.08
Test-S20-R31	20	FTRPD	10955	2791.8	2221	24.8	5	79.73
Test-S20-R32	20	FTRPD	10341	3382.1	3034	34.3	4	70.66
Test-S20-R33	20	FTRPD	10096	3086.1	2114	37.2	12	79.06
Test-S20-R34	20	FTRPD	9906	2759.6	2324	30.4	4	76.54
Test-S20-R35	20	FTRPD	10673	2821.2	2564	27.5	4	75.98
Test-S20-R36	20	FTRPD	7697	2647.1	2450	36.0	5	68.17
Test-S20-R37	20	FTRPD	12413	3174.3	2766	38.2	5	77.72
Test-S20-R38	20	FTRPD	15077	3225.8	1995	28.6	7	86.77
Test-S20-R39	20	FTRPD	9608	2774.4	2503	31.0	2	73.95
Test-S20-R40	20	FTRPD	11544	2956.2	2540	31.2	3	78.0
Test-S20-R41	20	FTRPD	11662	3262.9	2353	30.3	5	79.82
Test-S20-R42	20	FTRPD	12364	3181.7	2732	35.2	5	77.9
Test-S20-R43	20	FTRPD	9183	2780.5	1990	39.8	10	78.33
Test-S20-R44	20	FTRPD	8017	2891.5	2272	35.1	10	71.66
Test-S20-R45	20	FTRPD	13224	2942.0	2425	36.1	8	81.66
Test-S20-R46	20	FTRPD	8826	2815.5	2303	37.4	10	73.91
Test-S20-R47	20	FTRPD	10779	3159.2	2486	35.8	7	76.94
Test-S20-R48	20	FTRPD	12605	3250.9	2876	24.9	3	77.18
Test-S20-R49	20	FTRPD	11172	3540.0	2999	35.0	7	73.16
Test-S20-R50	20	FTRPD	12566	3009.9	2583	30.0	3	79.44

Table B.9: Test set size 20 results for TDRA with baseline parameters of drone speed 1.5 and coverage 25%.

Instance	c	Model	TRP	Average Cost	Best Cost	Run Time (sec)	Customers served by UAVs	Δ(%)
Test-S20-R1	20	FTRPD	11181	3034.3	2671	36.3	3	76.11
Test-S20-R2	20	FTRPD	12154	3230.1	2963	35.5	4	75.62
Test-S20-R3	20	FTRPD	10483	3202.3	2712	32.1	4	74.13
Test-S20-R4	20	FTRPD	10138	2793.1	2480	49.1	5	75.54
Test-S20-R5	20	FTRPD	10473	2681.8	2406	62.6	4	77.03
Test-S20-R6	20	FTRPD	10645	3047.6	2663	32.8	4	74.98
Test-S20-R7	20	FTRPD	15052	3653.9	3209	33.7	2	78.68
Test-S20-R8	20	FTRPD	11321	2935.3	2799	43.0	2	75.28
Test-S20-R9	20	FTRPD	12434	3190.1	2841	36.6	7	77.15
Test-S20-R10	20	FTRPD	10236	2749,9	2361	47.4	4	76.93
Test-S20-R11	20	FTRPD	9709	2726.8	2476	55.1	4	74.5
Test-S20-R12	20	FTRPD	11976	3082.1	2493	34.5	9	79.18
Test-S20-R13	20	FTRPD	10308	2977.0	2585	47.9	3	74.92
Test-S20-R14	20	FTRPD	8959	3171.9	2509	32.0	3	71.99
Test-S20-R15	20	FTRPD	11224	2624.6	2356	51.9	3	79.01
Test-S20-R16	20	FTRPD	12373	3318.3	3012	31.5	10	75.66
Test-S20-R17	20	FTRPD	10770	3090.0	2510	43.9	6	76.69
Test-S20-R18	20	FTRPD	9404	2961.4	2402	58.2	8	74.46
Test-S20-R19	20	FTRPD	9490	2871.6	2684	49.7	4	71.72
Test-S20-R20	20	FTRPD	13166	3890.8	2954	34.4	12	77.56
Test-S20-R21	20	FTRPD	11909	2576.0	2084	32.9	4	82.5
Test-S20-R22	20	FTRPD	10496	2935.5	2670	48.2	4	74.56
Test-S20-R23	20	FTRPD	12999	2887.9	2792	42.4	2	78.52
Test-S20-R24	20	FTRPD	10257	2859.3	2431	40.2	4	76.3
Test-S20-R25	20	FTRPD	9707	2565.1	2170	45.6	4	77.64
Test-S20-R26	20	FTRPD	10933	2836.3	2409	49.9	2	77.97
Test-S20-R27	20	FTRPD	11823	3310.5	2837	32.6	6	76.0
Test-S20-R28	20	FTRPD	9378	3157.8	2973	43.7	7	68.3
Test-S20-R29	20	FTRPD	9711	3359.2	3093	35.6	12	68.15
Test-S20-R30	20	FTRPD	11069	2779.8	2538	36.5	6	77.07
Test-S20-R31	20	FTRPD	10955	3011.0	2601	50.3	3	76.26
Test-S20-R32	20	FTRPD	10341	3057.6	2345	44.7	6	77.32
Test-S20-R33	20	FTRPD	10096	2902.3	2298	42.3	6	77.24
Test-S20-R34	20	FTRPD	9906	2811.6	2307	50.3	9	76.71
Test-S20-R35	20	FTRPD	10673	2873.9	2588	51.4	3	75.75
Test-S20-R36	20	FTRPD	7697	3247.3	2326	34.1	8	69.78
Test-S20-R37	20	FTRPD	12413	3174.7	2466	37.0	11	80.13
Test-S20-R38	20	FTRPD	15077	3507.1	2975	34.0	3	80.27
Test-S20-R39	20	FTRPD	9608	3200.9	2422	53.1	5	74.79
Test-S20-R40	20	FTRPD	11544	2681.1	2229	50.2	11	80.69
Test-S20-R41	20	FTRPD	11662	3113.8	2877	43.8	5	75.33
Test-S20-R42	20	FTRPD	12364	3129.4	2827	56.3	3	77.14
Test-S20-R43	20	FTRPD	9183	2636.9	2292	72.0	9	75.04
Test-S20-R44	20	FTRPD	8017	2973.1	2355	45.2	11	70.62
Test-S20-R45	20	FTRPD	13224	2959.6	2436	52.2	6	81.58
Test-S20-R46	20	FTRPD	8826	2986.7	2476	47.7	4	71.95
Test-S20-R47	20	FTRPD	10779	3430.1	2878	40.2	9	73.3
Test-S20-R48	20	FTRPD	12605	3358.0	3042	47.3	2	75.87
Test-S20-R49	20	FTRPD	11172	3636.0	3140	40.1	3	71.89
Test-S20-R50	20	FTRPD	12566	3178.0	2853	38.3	4	77.3

Table B.10: Test set size 20 results for DRLH with baseline parameters of drone speed 1.5 and coverage 25%.

B.2.3 Size 50

Table B.11 gives the detailed numerical results of TDRA on the test set instances of size 50. Column *Average Solution* reports the average of the best-found objective values for 10 runs on each instance.

Table B.12 gives the detailed numerical results of DRLH on the test set instances of size 50. Column *Average Solution* reports the average of the best-found objective values for 10 runs on each instance.

Instance	c	Model	TRP	Average Cost	Best Cost	Run Time (sec)	Customers served by UAVs	Δ(%)
Test-S50-R1	50	FTRPD	76040	18933.7	13865	310.3	4	81.77
Test-S50-R2	50	FTRPD	71479	20533.7	19717	398.9	9	72.42
Test-S50-R3	50	FTRPD	68991	15415.0	12039	163.6	13	82.55
Test-S50-R4	50	FTRPD	63106	16447.7	13703	309.7	2	78.29
Test-S50-R5	50	FTRPD	64344	18382.7	16665	370.9	8	74.10
Test-S50-R6	50	FTRPD	55808	15482.7	13610	316.1	2	75.61
Test-S50-R7	50	FTRPD	70996	14937.3	14185	179.6	3	80.02
Test-S50-R8	50	FTRPD	67045	15352.3	12189	221.9	2	81.82
Test-S50-R9	50	FTRPD	59370	18436.7	18249	317.6	4	69.26
Test-S50-R10	50	FTRPD	64735	15431.3	13282	203.0	2	79.48
Test-S50-R11	50	FTRPD	58336	13765.3	12392	239.6	5	78.76
Test-S50-R12	50	FTRPD	71199	18961.0	18152	275.0	5	74.51
Test-S50-R13	50	FTRPD	67789	15325.0	13705	191.0	2	79.78
Test-S50-R14	50	FTRPD	56062	17572.3	15582	435.3	6	72.21
Test-S50-R15	50	FTRPD	65911	15299.3	14861	386.6	6	77.45
Test-S50-R16	50	FTRPD	65473	13581.3	10715	311.9	3	83.63
Test-S50-R17	50	FTRPD	65093	21630.7	17632	534.9	31	72.91
Test-S50-R18	50	FTRPD	56543	16802.7	13607	259.9	3	75.94
Test-S50-R19	50	FTRPD	74598	19740.3	18544	349.9	8	75.14
Test-S50-R20	50	FTRPD	63204	16811.7	15705	327.3	3	75.15
Test-S50-R21	10	FTRPD	67226	22635.3	21628	412.8	11	67.83
Test-S50-R22	10	FTRPD	71793	16383.0	13591	279.3	3	81.07
Test-S50-R23	10	FTRPD	70201	13829.3	12645	218.5	2	81.99
Test-S50-R24	10	FTRPD	61947	15006.7	13047	219.1	3	78.94
Test-S50-R25	10	FTRPD	69611	15661.3	12778	231.2	3	81.64
Test-S50-R26	10	FTRPD	60856	17520.0	15805	325.1	6	74.03
Test-S50-R27	10	FTRPD	58800	14893.0	13419	215.0	3	77.18
Test-S50-R28	10	FTRPD	65007	18289.0	16701	352.0	4	74.31
Test-S50-R29	10	FTRPD	65158	17760.3	17262	330.3	5	73.51
Test-S50-R30	10	FTRPD	73556	19000.7	14435	352.1	5	80.38
Test-S50-R31	10	FTRPD	77752	17607.7	16266	284.0	5	79.08
Test-S50-R32	10	FTRPD	62046	15936.7	12639	429.7	10	79.63
Test-S50-R33	10	FTRPD	65750	18995.0	15819	324.3	4	75.94
Test-S50-R34	10	FTRPD	59228	18103.0	15497	277.6	4	73.84
Test-S50-R35	10	FTRPD	70543	19793.7	18361	290.1	3	73.97
Test-S50-R36	10	FTRPD	63075	13761.0	11624	260.9	4	81.57
Test-S50-R37	10	FTRPD	63480	15479.0	14071	333.3	6	77.83
Test-S50-R38	10	FTRPD	79436	17151.0	15006	252.8	5	81.11
Test-S50-R39	10	FTRPD	67564	19722.3	15505	309.8	3	77.05
Test-S50-R40	10	FTRPD	74127	13893.7	11076	294.4	2	85.06
Test-S50-R41	10	FTRPD	62241	19883.0	16016	256.0	3	74.27
Test-S50-R42	10	FTRPD	63729	14176.3	13227	265.9	6	79.24
Test-S50-R43	10	FTRPD	55776	14852.7	14225	297.2	6	74.5
Test-S50-R44	10	FTRPD	72291	15261.7	14426	260.8	2	80.04
Test-S50-R45	10	FTRPD	60357	16808.7	11681	321.6	4	80.65
Test-S50-R46	10	FTRPD	55791	13305.0	12456	145.1	2	77.67
Test-S50-R47	10	FTRPD	63701	16112.0	13123	278.7	3	79.4
Test-S50-R48	10	FTRPD	59799	15152.0	12869	337.5	5	78.48
Test-S50-R49	10	FTRPD	69990	13218.7	12986	162.7	5	81.45
Test-S50-R50	10	FTRPD	69061	12683.3	11958	225.9	3	82.68

Table B.11: Test set size 50 results for TDRA with baseline parameters of drone speed 1.5 and coverage 25%.

Instance	c	Model	TRP	Average Cost	Best Cost	Run Time (sec)	Customers served by UAVs	Δ(%)
Test-S50-R1	50	FTRPD	76040	16797.8	13769	237.4	2	81.89
Test-S50-R2	50	FTRPD	71479	19148.0	15856	346.4	4	77.82
Test-S50-R3	50	FTRPD	68991	16689.0	12909	321.5	2	81.29
Test-S50-R4	50	FTRPD	63106	14761.8	13446	336.6	2	78.69
Test-S50-R5	50	FTRPD	64344	16855.8	13836	390.5	4	78.50
Test-S50-R6	50	FTRPD	55808	16039.5	14410	265.1	2	74.18
Test-S50-R7	50	FTRPD	70996	16840.0	15287	281.3	3	78.47
Test-S50-R8	50	FTRPD	67045	13910.0	13286	194.4	2	80.18
Test-S50-R9	50	FTRPD	59370	17897.0	16757	338.5	7	71.78
Test-S50-R10	50	FTRPD	64735	14479.8	14270	369.5	2	77.96
Test-S50-R11	50	FTRPD	58336	14675.3	13683	348.6	2	76.54
Test-S50-R12	50	FTRPD	71199	15867.5	14433	315.0	4	79.73
Test-S50-R13	50	FTRPD	67789	15839.5	15402	331.3	5	79.78
Test-S50-R14	50	FTRPD	56062	15628.8	14031	435.3	6	74.97
Test-S50-R15	50	FTRPD	65911	16792.0	13956	297.6	2	78.83
Test-S50-R16	50	FTRPD	65473	16441.3	10715	404.9	3	83.63
Test-S50-R17	50	FTRPD	65093	20502.5	18962	386.4	7	70.87
Test-S50-R18	50	FTRPD	56543	14973.0	14124	336.5	3	75.02
Test-S50-R19	50	FTRPD	74598	15632.8	14461	196.4	2	80.61
Test-S50-R20	50	FTRPD	63204	16431.5	13760	280.8	2	78.23
Test-S50-R21	50	FTRPD	67226	17725.4	15363	331.9	3	77.15
Test-S50-R22	50	FTRPD	71793	16049.6	14343	297.1	2	80.02
Test-S50-R23	50	FTRPD	70201	16956.0	15316	264.9	3	78.18
Test-S50-R24	50	FTRPD	61947	16064.0	15415	319.1	6	75.12
Test-S50-R25	50	FTRPD	69611	16159.2	15136	397.2	3	78.26
Test-S50-R26	50	FTRPD	60856	17729.7	16259	364.9	7	73.28
Test-S50-R27	50	FTRPD	58800	15694.2	13956	473.4	6	76.27
Test-S50-R28	50	FTRPD	65007	17102.0	15304	391.6	4	76.46
Test-S50-R29	50	FTRPD	65158	16810.4	14988	344.8	2	77.00
Test-S50-R30	50	FTRPD	73556	15815.2	13826	537.3	2	81.20
Test-S50-R31	50	FTRPD	77752	15837.6	14821	321.9	2	80.94
Test-S50-R32	50	FTRPD	62046	15476.0	14393	493.7	2	76.80
Test-S50-R33	50	FTRPD	65750	17253.2	14606	452.1	5	77.79
Test-S50-R34	50	FTRPD	59228	14623.2	13921	451.8	2	76.50
Test-S50-R35	50	FTRPD	70543	18341.0	16787	316.2	3	76.20
Test-S50-R36	50	FTRPD	63075	15187.8	13840	418.2	2	78.06
Test-S50-R37	50	FTRPD	63480	15938.2	14437	530.6	2	77.26
Test-S50-R38	50	FTRPD	79436	16685.8	15910	475.3	7	79.97
Test-S50-R39	50	FTRPD	67564	17202.8	14510	322.1	2	78.52
Test-S50-R40	50	FTRPD	74127	16699.5	15528	545.3	5	79.05
Test-S50-R41	50	FTRPD	63729	14813.0	14267	458.4	2	77.61
Test-S50-R42	50	FTRPD	63729	14871.5	14267	458.4	2	77.61
Test-S50-R43	50	FTRPD	55776	18252.6	13971	497.7	5	74.95
Test-S50-R44	50	FTRPD	72291	16086.0	15319	343.4	2	78.81
Test-S50-R45	50	FTRPD	60357	14365.2	13815	424.6	2	77.11
Test-S50-R46	50	FTRPD	55791	15738.3	14134	410.1	2	74.67
Test-S50-R47	50	FTRPD	63701	15413.8	14124	477.7	2	77.83
Test-S50-R48	50	FTRPD	59799	15250.0	13823	375.5	2	76.88
Test-S50-R49	50	FTRPD	69990	17357.3	15266	389.5	5	78.19
Test-S50-R50	50	FTRPD	69061	16148.8	14455	536.3	2	79.07

Table B.12: Test set size 50 results for DRLH with baseline parameters of drone speed 1.5 and coverage 25%.