University of Bergen
Department of Informatics

Optimizing Feeder Network Design with Deep Reinforcement Learning: A Hyperheuristic Approach

Author: Fredrik Nestvold Larsen
Supervisors: Ahmad Hemmati

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

July, 2023

Abstract

Across the globe, hundreds of shipping networks form an intricate web of trade routes forming the backbone of international commerce. These networks are responsible for an estimated 80 percent of all cargo transported globally and are known as Liner Shipping Network Design Problem (LSNDP) in the literature. This thesis will focus on a variant of the LSNDP known as the feeder networks. It is the problem of serving a number of shipping requests using a fleet of vessels. Each request involves moving a number of containers from the origin port to the destination port. Our objective is to design routes that connect all ports in the most optimized order such that pickup and deliveries correspond with the lowest cost possible. We will implement, adapt and compare two state-of-the-art frameworks, where one (Adaptive Heuristic) framework is optimized and created for the FNDP while the other (Deep Reinforcement Learning Hyperheuristic) is a more general framework for a multitude of different Combinatorial Optimization Problems.

Acknowledgements

First and foremost, I would like to thank my supervisor Ahmad Hemmati for his guidance and continuous help throughout the master project. I also want to express my heartfelt gratitude to my family, friends, and especially my girlfriend for their unwavering support and encouragement throughout this journey. Their presence and belief in me have kept me motivated and focused throughout this last year. Lastly, I want to acknowledge and appreciate the efforts of my fellow Machine Learning students. The opportunity to learn, grow, and exchange ideas with such talented individuals has been invaluable. This journey would not have been the same without you.

`Fredrik Nestvold Larsen`
Saturday 1st July, 2023

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Liner shipping, often considered the backbone of global supply chains, involves the transportation of tens of thousands of containers on fixed routes using over 7,000 container and roro ships and over 1,900 liner shipping services worldwide. These services provide regular and scheduled transportation between ports, facilitating the loading and unloading of containers at seaports globally, connecting ports worldwide in an intricate web of trade. (wor).

The challenge of creating networks of services for these container ships gives rise to complex planning problems, such as the Liner Shipping Network Design Problem (LSNDP). This thesis focuses a specific variant of the LSNDP, the Feeder Network Design Problem (FNDP) with the possibility of hub-and-spoke structure within the network. The FNDP involves designing feeder networks that revolve around a single hub port and several feeder ports, aiming to transport several containers either originating from or destined for the hub port. These demands are known in advance, and the task is to decide on a set of routes and a fleet of vessels that transport the containers between the hub port and the feeder ports on a weekly schedule.

We developed a hyperheuristic based on the Deep Reinforcement Learning Hyperheuristic (DRLH) framework by Kallestad et al. (2023) and also a variant of the Adaptive Heuristic (AH) by Bergmann et al. (2023) framework to address this complex challenge of optimizing feeder networks.

1.1 Context and Motivation

Globalization has led to an exponential increase in the complexity and scale of shipping networks, thereby intensifying the importance of optimal planning and scheduling. The FNDP is at the heart of this logistical challenge, representing an essential component of this global process, with the efficient design of these feeder networks acting as a crucial determinant of overall supply chain performance.

The FNDP is traditionally approached through various metaheuristics, owing to the high-dimensionality and non-linearity of the problem. However, these classical methods often require domain knowledge and extensive computational resources. Moreover, they may fail to adapt effectively to dynamic changes in the system, such as variations in container demand or port availability. On the other hand, Deep Reinforcement Learning (DRL) is an emerging field of machine learning that demonstrates promise in addressing complex optimization problems. DRL can handle large-scale, dynamic problems and learn from its environment over time. However, the application of DRL in the feeder network domain remains relatively unexplored. Given these circumstances, the motivation of this thesis lies in analyzing the performance of a powerful metaheuristic - the Adaptive Heuristic - against an adapted version of Deep Reinforcement Learning Hyperheuristic (DRLH) in solving the FNDP. This study aims to provide insights into the strengths and weaknesses of these two heuristic approaches, to potentially derive a hybrid that incorporates the advantages of both.

1.2 Thesis Outline

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

Chapter 2 - Background and Related Work gives the theoretical background related to combinatorial optimization, deep learning, and reinforcement learning required for this thesis. It also covers related works and different solution methods previously utilized to solve combinatorial optimization problems.

Chapter 3 - Problem Sets introduces two different versions of the Feeder Network Design Problem. This gives the necessary knowledge to understand better what we are trying to solve. It goes deeper into different structures, restrictions, and conditions of the problem at hand.

Chapter 4 - Adaptive Heuristic introduces the latest state-of-the-art metaheuristic used to solve FNDP. It offers a detailed overview of the framework, helping to understand its workings better and giving insight into the heuristics.

Chapter 5 - DRLH introduces the Deep Reinforcement Learning Hyperheuristic model that we use in this thesis. It offers a detailed overview of the framework, helping to understand its workings better.

Chapter 6 - Experimental Setup contains the specifics of the environment the experiments were conducted. This includes hardware, baseline, parameters for the Adaptive Heuristic and hyperparameters for DRLH, and the generation of the training data utilized by DRLH.

Chapter 7 - Results presents the findings of the thesis and discusses their relevance and impact.

Chapter 8 - Conclusion and Future Work summarizes and concludes the work and float some ideas for future work related to this thesis.

Chapter 2

Background and Related work

The purpose of this background section is to provide the necessary foundational information to understand the research conducted in this study. It covers the essential concepts and components that are crucial for comprehending the core objectives and findings of the research. By exploring this background, readers will understand the fundamental elements that form the basis of the research and its importance within the broader scientific context.

2.1 Combinatorial Optimization Problems

Optimization is the science of making the best possible decision. Whether we seek a minimum or a maximum, the goal remains the same: identify the solution that best satisfies the predetermined criteria. A subset of optimization, Combinatorial Optimization (CO), applies this concept to finite, often discrete structures. The goal is to find the optimal object from a finite set of objects (Schrijver, 2003).

Combinatorial Optimization Problem (COP) revolves around the goal of optimally allocating finite resources, which can only be separated into distinct, indivisible units, to maximize or minimize a specific objective. These problems are often challenging to solve using traditional methods, and their complexity increases with the number of variables or constraints. COPs are present in various domains, such as logistics, transportation, scheduling, and finance. Examples of COPs are but not limited to Scheduling Problem (SP), Knapsack Problem (KP), Vehicle Routing Problem (VRP) and Minimum Spanning Tree (MST).

2.1.1 Liner Shipping Network Designs Problem

Liner Shipping Network Design Problem (LSNDP) involves creating optimal plans for the routes of container ships and the allocation of cargo, considering operational constraints and business requirements. This type of problem aims to minimize the network's objective costs, such as fuel consumption, port charges, and transit time, while ensuring service frequency and connectivity among ports. In LSNDP, the most common unit of measurement is the Twenty-foot equivalent unit (TEU), a standard size for shipping containers. Some studies (Brouer et al., 2014) may also use the less common Forty-foot equivalent unit (FEU), which is equal in length to two TEUs, allowing for efficient stacking of the containers.

Research in this field has been divided into four categories (Meng et al., 2014), as illustrated in Figure 2.1.

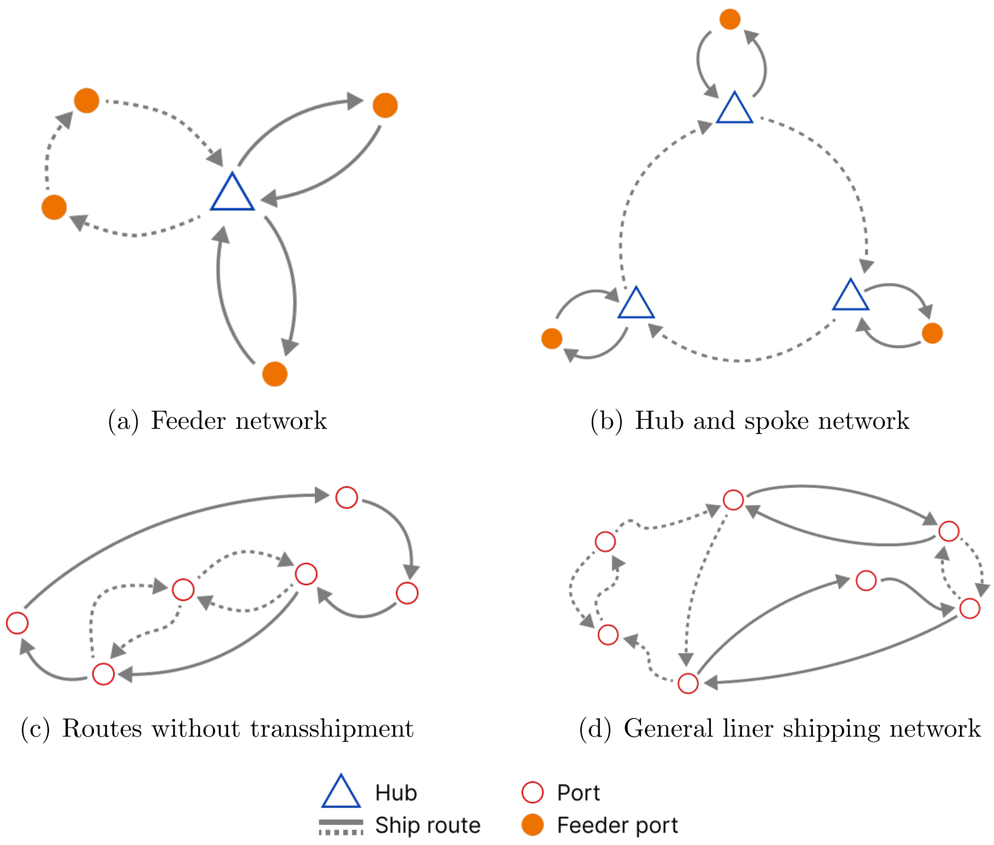

(a) Feeder network
(b) Hub and spoke network
(c) Routes without transshipment
(d) General liner shipping network
Hub
Port
Ship route
Feeder port

Figure 2.1: Overview of liner shipping network categories

LSNDPs are complex and computationally challenging problems that have been studied using operations research and optimization models. They are known to be NP-hard problems, meaning that they are at least as hard as the hardest problems in NP (non-deterministic polynomial time) class, and there is no known algorithm to solve them efficiently.

2.2 Solution Methods

Various methodologies are available for resolving COPs. For a better understanding, these approaches can be categorized into two primary groups, *exact approaches* and *heuristic approaches*. Although this thesis primarily focuses on heuristic approaches and their various categories, it is essential to note that exact approaches are also available as an alternative method for solving COPs. To provide context for the alternative methods, we will briefly explain what constitutes an exact approach when solving COPs.

2.2.1 Exact Approach

The main objective of the exact approach is to find the globally optimal solution for the problem. This approach involves exploring the entire search space to ensure that the solution found is the best possible solution based on the constraints and objective function of the problem. Although this can result in high run times, which scale with the complexity and size of the problem, it guarantees an optimal solution for the given problem.

2.2.2 Heuristic Approach

The heuristic approach aims to find an approximate solution to a combinatorial problem in a reasonable amount of time without guaranteeing to find the optimal solution. However, the solutions found are usually satisfactory to the specific use case of the search. Compared to exact methodologies, heuristic strategies often prove faster and become the go-to alternatives for large-scale instances where pinpointing the globally optimal solution is not feasible. Over several decades, this has led to the popularity and extensive study of heuristics within the optimization community.

Two main ways of conducting a search exist when using the heuristic approach: *constructive heuristics* and *perturbative heuristics* (also known as local search heuristics). While the former involves systematically assembling a solution from scratch, the latter focuses on enhancing an existing solution through targeted modifications. In the following sections, we will examine each of these methodologies in greater detail.

Constructive

A constructive heuristic is a type of approach that incrementally builds a solution, adding one element at a time until the solution is complete. Compared to random strategies, this method generally yields better results. However, it may struggle to match the performance of exact or perturbative heuristic methods. One advantage of constructive heuristics is their speed—they produce solutions swiftly. As a result, they are frequently utilized to create initial feasible solutions.

Perturbative

The power of perturbative heuristics lies in their ability to navigate through the solution space effectively, balancing between exploring new areas (*diversification*) and improving the best-found solutions (*intensification*).

A perturbative heuristic is an approach that makes alterations to an existing solution (s) to generate a new solution (s'). This form of heuristic is widely used in optimization research. These heuristics can be universally applicable to a range of different problems or specifically tailored to a unique problem, depending on the problem distinct characteristics and constraints. A significant aspect of perturbative heuristics involves the concept of a *neighborhood*. The *neighborhood* of a solution includes all potential solutions that can be attained by making small *modifications* or *moves* to the current solution using the heuristic. These moves typically involve *swapping*, *inserting*, or *removing* elements in the solution. This neighborhood concept is crucial in controlling how much the solution should be changed and exploring the solution space efficiently.

When using a perturbative heuristic, it is common practice to create a pool of several heuristics rather than relying on a single one. This tactic often leads to better results, enabling a broader and more diverse exploration of the solution space and offering more opportunities to escape local optima and reach global optima. This strategy of combining multiple heuristics was shown to yield better results as early as the work of Fisher and Thompson (1963).

2.2.3 Metaheuristics

A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms (Sörensen and Glover, 2013). The beauty of metaheuristics is that they are not tailored to a specific problem. Instead, they are general-purpose strategies that can be applied to a wide range of problems.

Metaheuristics can be characterized by their ability to balance between exploration (*diversification*) and exploitation (*intensification*) in the search process (i.e., efficiently exploring the solution space of the problem). Metaheuristics are strategies that "go beyond" standard heuristics to find solutions to challenging optimization problems. Popular metaheuristics include Genetic Algorithm (GA), Simulated Annealing (SA), Tabu Search (TS), and Ant Colony Optimization (ACO), among others. Diverse phenomena, such as biological evolution, thermodynamics, human memory, and the behavior of ant colonies, have inspired these techniques. Each of these techniques uses a different strategy. Still, they all aim to find a near-optimal solution to optimization problems where the exact solution is often hard or impossible to find within a reasonable time frame.

Adaptive Large Neighborhood Search

Adaptive Large Neighborhood Search (ALNS) is a metaheuristic framework frequently used to tackle complex COPs. ALNS was first introduced by Ropke and Pisinger (2006) and is an extension of the Large Neighborhood Search (LNS) framework of Shaw (1998). ALNS works by repeatedly exploring a broad neighborhood of possible solutions, then adaptively fine-tuning the search based on the solutions' quality. To achieve this, ALNS uses a set of predefined heuristics, which perform various modifications on the solution. These modifications might include *removal*, *insertion*, *deletion*, *swap*, or any other suitable change depending on the problem. During the search, ALNS evaluates these operators based on their effectiveness in improving the solution or leading to less desirable outcomes. It then dynamically chooses the operators for use based on these evaluations, giving preference to those operators that have consistently led to better solutions.

2.3 Hyperheuristic

While traditional heuristics solve problems by directly finding solutions, hyperheuristics work differently. They take a step back, and instead of looking for solutions themselves, they look at how to choose or create the best method (or heuristic) to find these solutions. This 'meta' approach provides hyperheuristics with a versatile edge, enabling their application across a diverse array of problem types.

The concept of a *hyperheuristic* was first mentioned in 1997, but it was not used in the context of combinatorial optimization until 2001 by Cowling et al.. Cowling et al. characterized a hyperheuristic as "heuristics to choose based on the characteristics of the region of the solution space currently under exploration." This definition was expanded upon by Burke et al. in 2010, who proposed that a hyperheuristic is "a search method or learning mechanism for selecting or generating heuristics to solve computational search problems." Burke et al. (2010) also state "We consider a hyperheuristic to be a learning algorithm when it uses some feedback from the search process," which makes it very applicable to Reinforcement Learning.

2.4 Reinforcement Learning

Machine learning is traditionally categorized into *Supervised Learning*, *Unsupervised Learning*, and *Reinforcement Learning (RL)*. Each category holds its unique value and serves distinct purposes in various contexts.

Supervised Learning operates on labeled datasets where each input corresponds to a specified output. During training, the model receives an input and generates an output. Based on the comparison with the actual output (label), the model is instructed on how to improve. The goal is for the model to generalize from this experience so it can make accurate predictions on unseen data. This ability to predict unseen data is the true strength of Supervised Learning.

Unsupervised Learning is a powerful approach that focuses on analyzing unlabeled data to unveil inherent structures, patterns, and relationships. Instead of making predictions, its purpose is to gain insights into the underlying structure and patterns within the data. This methodology is particularly valuable in situations where labeled data is limited or unavailable or when the primary objective is to comprehend the inherent characteristics of the data.

Reinforcement Learning (RL) differs significantly from both Supervised and Unsupervised Learning. The key concept of RL revolves around an *agent* that interacts with an *environment.* Its fundamental goal is to accumulate as much reward as possible. Doing so optimizes a *policy*, essentially the decision-making process that dictates the *agent's* actions. The uniqueness of RL lies in its trial-and-error approach, where the agent learns which actions are beneficial in specific situations through environmental interaction. There are no labels to guide the agent. The only feedback it receives is the *reward signal* from the *environment*, which shifts over time based on the *agent's* experiences and interactions.

The types of problems that RL tackles are those in which the underlying model of the environment is unknown and is affected by the agent's choice of actions Sutton (2018).

2.4.1 Agent-Environment Interface

The *Agent-Environment Interface* is a fundamental concept in RL that illustrates the interaction between the agent and the environment. This interface separates the agent's decision-making process from the environment's dynamics, see Figure 2.2.

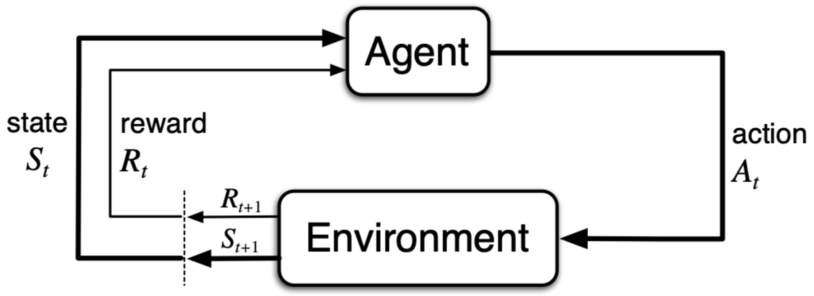


Figure 2.2: Agent interaction w/ environment

Credit: Sutton and Barto, "An introduction to Reinforcement Learning", 2018

At each time step, the agent observes the current state of the environment, chooses an action based on its policy, receives a reward from the environment, and transitions to a new state. The agent fine-tunes its policy through repeated interactions with the environment, improving its ability to select actions that increase the cumulative reward.

Environment

The *environment* in RL is essentially a simulation of the problem to be solved. It can take a variety of forms - it could be a virtual representation of a city for a self-driving car, or it could be something more abstract, such as a mathematical problem in our case. The environment plays a crucial role in the learning process, as it defines the dynamics of the problem, including what actions are possible and what constraints exist. It responds to the agent's actions by presenting a new state and a corresponding reward, guiding the agent's learning trajectory.

Agent

The *agent* is the entity that observes its surroundings, learns from the feedback it receives, and makes decisions accordingly. Positioned within an environment, the agent perceives the current situation or *state* and executes an *action* based on these observations. After taking an action, the agent receives feedback from the environment in the form of a reward and a new state. This feedback serves as valuable information for the agent, providing insights into the outcomes of its actions and shaping its learning process. The agent's primary objective is to maximize the total accumulated reward over time, thereby acquiring the ability to make optimal decisions throughout the learning process.

Reward Signal

The *reward signal* (denoted r) is a vital part of the feedback loop in RL. It is a signal that the agent receives from the environment after each action, which informs the agent about the quality or effectiveness of the action executed. The agent's primary goal is to learn how to maximize the total reward over time. In other words, the agent wants to make decisions that lead to the highest possible rewards in the long run. The reward signal sets the objective for the RL problem. It is also worth noting that the design of the reward signal is essential, as it can directly influence the agents learning trajectory and behavior.

Policy

A policy (denoted π) in RL can be viewed from several perspectives. However, Sutton (2018) offers a straightforward definition by describing the policy as "the agent's chosen method of behavior at a given time." In more concrete terms, the policy gives us the probability distribution $\pi(a_t|s_t)$, which represents the likelihood of the agent taking action a from the action set $\mathcal{A}(s_t)$ when in state s_t from the state space $\mathcal{S}$. The ultimate goal in RL is to discover an optimal policy (denoted as π_*), which maximizes the expected cumulative reward over time for each state, thus ensuring the best long-term payoff for the agent. It is important to note that policies can be deterministic, returning one action, or stochastic, which provides a probability distribution over all possible actions.

Value function

The concept of the *value function* $v_\pi(s)$ plays a crucial role in estimating the desirability or goodness of being in a specific state for an agent. It measures the expected cumulative reward that an agent can obtain starting from a particular state and following a given policy π. By estimating the value function, the agent gains insights into the long-term benefits and potential outcomes of being in different states, allowing it to make informed decisions and navigate the environment more effectively.

2.4.2 RL methods

RL approaches can be categorized into value-based, policy-based, and actor-critic methods, as defined by Sutton (2018). These approaches differ in the following key aspects:

- **Value-based methods:** These approaches focus on estimating and optimizing the value function, such as the state-value or action-value functions. They aim to determine the value of different states or state-action pairs and derive an optimal policy based on these value estimates.
- **Policy-based methods:** Unlike value-based methods, policy-based approaches directly learn and improve the policy function without explicitly estimating the value function. They explore different policies and update the policy parameters to maximize the expected cumulative reward.

- **Actor-critic methods:** Actor-critic methods combine elements of both value-based and policy-based approaches. They maintain an actor, responsible for selecting actions based on a policy, and a critic, which evaluates the actions the actor takes using value-based methods. This combination allows for more stable and efficient learning by utilizing the strengths of both approaches.

2.5 Deep Learning

Deep Learning (DL) is a powerful and rapidly advancing subset of machine learning that has revolutionized various fields, from computer vision to natural language processing. At its core, DL involves using neural networks, which are complex mathematical models inspired by the structure and function of the human brain. These networks are designed to learn patterns and relationships within large datasets, enabling them to make predictions, classifications, and **decisions** with remarkable accuracy and speed.

DL has become increasingly popular in recent years due to the explosion of big data and the availability of powerful computing resources. By leveraging these resources, DL models can be trained on big amounts of data to identify complex patterns and relationships that would be impossible for humans to detect. This has led to breakthroughs such as image recognition, speech recognition, and autonomous driving. In recent years, advances in DL have led to the development of deep reinforcement learning, which combines RL with neural networks to enable agents to learn from high-dimensional inputs such as images or speech. This has expanded the range of applications of RL and opened up new research directions.

Policy gradient methods, a subset of policy search methods from traditional RL, use neural network weights to encode the policy $\pi(a|s, \theta)$ parameter θ. As a result, optimizing the parameters of θ to discover the optimal policy equates to optimizing the neural network's weights. This is advantageous because deep learning techniques, like back-propagation, can be utilized to find the optimal policy. Due to their network-based structure, deep RL methods have been highly effective in addressing RL tasks. One of the key benefits of policy gradient methods is their stable convergence property, which ensures steady improvement at each time step. This stands in stark contrast to value-based methods, where updates to the value function can trigger dramatic behavior changes, resulting in considerable oscillations during training. Moreover, policy gradient methods perform well under conditions of uncertainty, as they are adept at learning stochastic policies. However, they also have a known disadvantage: they are prone to converge to local optima, as opposed to the global optimum (Sutton (2018)).

2.5.1 Proximal Policy Optimization

Proximal Policy Optimization (PPO) was first introduced by Schulman et al. (2017) from OpenAi. PPO has had a significant impact on the field of RL. The PPO algorithm was designed to address the challenges associated with other policy optimization methods, such as Policy Gradient (PG) and Actor-Critic (AC) methods in RL, particularly the issue of harmful, large policy updates. By creating a more cautious approach to policy updates, PPO ensures a more stable learning process and prevent *policy collapse*, where policy performance degrades significantly and never recovers.

Priniciple of PPO

The underlying principle of PPO is limiting the policy update at each iteration to ensure that the learning process is more stable. PPO modifies the objective function rather than directly optimizing the expectation of the sum of future rewards as in conventional policy optimization. This modification results in a new 'surrogate' objective that bounds the policy update, thus preventing overly large and potentially destabilizing updates.

Mathematical framework

The variable r_t signifies the ratio of probabilities between the current policy, π_θ, and the prior policy, $\pi_{\theta_{old}}$, as shown in $r_t = \dfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \tag{2.1}$$

Here, $\hat{A}_t$ represents the advantage. The *clip* operation is used to restrict the value of $r_t(\theta)$ within the boundary $[1-\epsilon, 1+\epsilon]$. This implies a constrained relationship between the present and preceding policy outcomes, thereby moderating the variance in the behavior exhibited by the current policy as compared to its preceding counterpart.

2.6 Related Work

Bergmann et al. (2023) proposed a metaheuristic framework that tackled two variants of the Feeder network design, which incorporates a general origin-destination pattern and speed optimization. Performance is compared against an Adaptive Large Neighborhood Search framework on instances from the literature and outperformed the previous framework. To evaluate the effectiveness of the AH, computational experiments were conducted on various problem instances. These instances were derived from the adapted LINER-LIB benchmark suite Brouer et al. (2014), adapted from previous studies, and generated in-house. The results demonstrated that AH outperformed the exiting results in terms of solution quality and computation time.

Kallestad et al. (2023) proposed a general hyperheuristic framework named Deep Reinforcement Learning Hyperheuristic, demonstrating its efficacy when applied to a select range of COPs. In order to assess the framework's performance, the researchers conducted computational experiments on an assortment of problem instances. They then compared the outcomes with those from the Adaptive Large Neighborhood Search (ALNS) and a straightforward Uniform Random Sampling (URS) framework. The results indicated that the proposed hyperheuristic framework consistently produced superior solutions.

Chapter 3

Problem Sets

The Feeder Network Design Problem (FNDP) revolves around the optimization of feeder networks to balance costs and efficiency in liner shipping. The goal is to maximize profits while minimizing expenses, which often involves using large vessels, known as mother vessels, capable of carrying substantial amounts of cargo – up to 18,000 TEU (Twenty-foot equivalent unit) or 9,000 FEU (Forty-foot equivalent unit). However, the use of such large vessels introduces its own set of challenges. While some major ports, such as Shanghai Port and Singapore Port, are equipped to handle these colossal ships, many smaller ports lack the necessary infrastructure. Thus, these smaller ports are unable to accommodate mother vessels, primarily due to their size. Additionally, liner shipping networks regularly visit all ports within their network while adhering to specific time windows and constrictions. This complex balancing act necessitates the use of smaller vessels, known as daughter vessels, to service parts of the network. By employing daughter vessels, we can maximize the utilization of each ship, thereby minimizing costs and emissions. This approach ultimately results in an optimized network tailored to the specific set of ports. In this chapter, we consider two variants of Feeder Networks and include mathematical models to define the problems better.

3.1 Feeder Network Design Problem with Optional Transshipment

While FNDP concerns planning direct transport routes from a central hub to various locations, Feeder Network Design Problem with Optional Transshipment (FNDP-OT) introduces the flexibility of transferring demands between vessels along the route for increased efficiency and adaptability as shown in Figure 3.2. The FNDP-OT requires designing a network of routes for a given hub (continental port), a set of feeder ports, and a number of vessel types to ensure the weekly demand of all ports is met. Additionally, it involves determining the fleet of vessels required to service these routes. Each port has a certain number of containers, usually measured in TEUs, that must be transported to and from the hub. These container movements are known as pickups and deliveries. The vessels are subject to maximum capacity, the maximum number of TEUs they can carry, and a weekly time charter rate (TC rate) and fuel consumption. The TC rate includes the cost of operating a given vessel for a given time period, excluding the fuel consumption incurred while sailing between ports. In this thesis, we consider the same version of the FNDP as Variant 2b, as proposed by Msakni et al. (2020), which allows for *hub-and-spoke* structures within the feeder network.

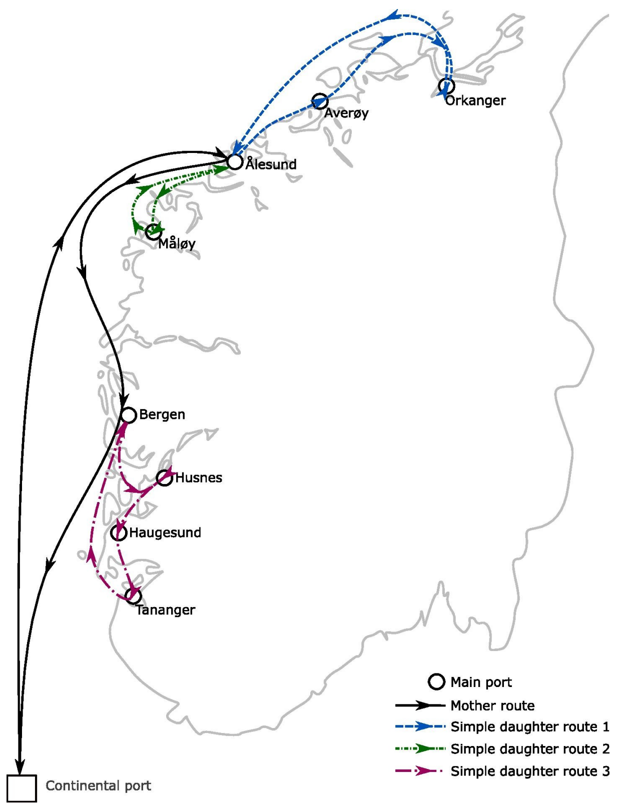


Figure 3.1: A feasible solution for a Feeder Network with Optional Transshipment along the Norwegian coast. Effectively managing cargo shipment with one mother route and three daughter routes. Credit: Msakni et al. (2020).

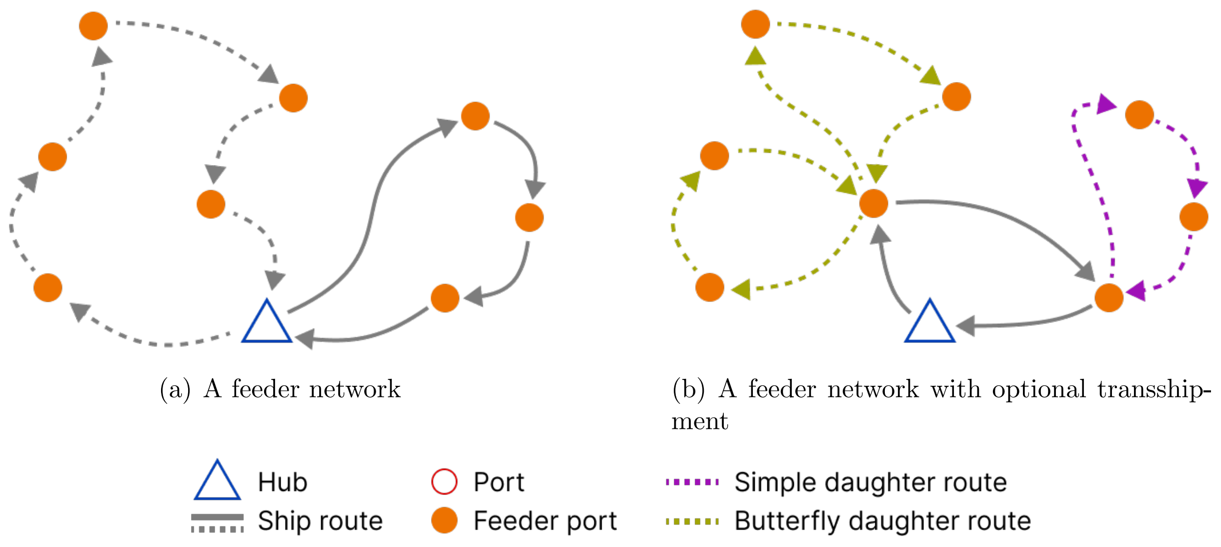
To achieve this *hub-and-spoke* structure within the network, we consider *mother and daughter routes*. A *mother route* originates and ends at the *hub* while visiting a subset of ports in between. *Daughter routes*, on the other hand, are routes that originate and end at a *feeder port* (i.e., a *transshipment port*) and visit a subset of ports in between. However, the *daughter routes* cannot include the *hub*. Therefore there must exist at least one *mother route* to activate a *transshipment port* from a *feeder port* for *daughter routes* to be enabled. Any feeder port may be visited once, with the exception of transshipment ports, where one *mother route* and multiple *daughter routes* may visit. Different types of vessel service mother and daughter routes. We call these mother vessels and daughter vessels. Due to the enabling of daughter routes and the fact that they are only optional, we denote this version of the problem as Feeder Network Design Problem with Optional Transshipment (FNDP-OT).

Transshipment of cargo

In the context of liner shipping, transshipment involves moving cargo from one ship to another, typically at a transshipment or feeder port. Transshipment within the feeder network reduces costs, improves connectivity, and enhances scalability.

3.1.1 Daughter route structures

The left network in Figure 3.2 illustrates a possible solution for a conventional feeder network consisting solely of mother routes, while network on the right is a possible solution for a feeder network that allows hub-and-spoke structure, this is made possible with the inclusion of daughter routes.



(a) A feeder network

(b) A feeder network with optional transshipment

Figure 3.2: Feeder Network Structures

In this thesis, two kinds of daughter routes are to be considered: *simple routes* and *butterfly routes*, and establish a one-week (7 days) time window for both types of daughter routes based on practical considerations.

A *simple route* (the purple route in Figure 3.1) is a round trip that begins and ends at the same *feeder port*, visiting a subset of other ports in between. It is a straightforward, efficient route plan that allows the daughter vessel to load, transport, and unload cargo in one continuous journey.

A *butterfly route* (the yellow route in Figure 3.2), by contrast, is essentially a concatenation of two simple routes, starting and ending at the same *feeder port*. The necessity for butterfly routes arises when a daughter vessel's capacity constraints do not allow it to transport all the cargo in one run. Instead, the vessel makes two distinct loops, visiting a set of ports on each loop and returning to the origin feeder port in between. Despite this additional complexity, the butterfly route must still fit within the one-week time window. It is worth noting that this two-tiered structure of simple and butterfly routes within daughter routes provides a balance of simplicity and flexibility. It allows the network to adapt to varying cargo demands while maintaining the efficiency of operations within the established time constraints.

3.1.2 Mathematical Model

The mathematical framework for the FNDP-OT is an updated model by Msakni et al. (2020). We include it to provide a clear mathematical representation of the problem. Definitions of various sets, parameters, and decision variables are introduced as follows:

Sets

$\mathcal{K}$	set of available mother vessel types (in TEUs)
$\mathcal{R}^{\mathcal{M}}$	set of candidate mother routes
$\mathcal{R}^{\mathcal{M}}_p$	set of candidate mother routes that include feeder port p
$\mathcal{R}^{\mathcal{D}}$	set of ports that precede the visit of port p in mother route r
$\mathcal{R}^{\mathcal{D}}_{rp}$	subset of daughter routes that can do transshipment at port p with mother route r

$\mathcal{P}_r$	set of served feeder ports by mother route r
$\mathcal{P}^-_{rp}$	set of ports that precede the visit of port p in mother route r
$\mathcal{D}^{\mathcal{R}}$	set of route pairs where a pair indicates which daughter route can be connected to which mother route i.e., $\mathcal{D}^{\mathcal{R}} = \{(r,d), r \in \mathcal{R}^{\mathcal{M}}, p \in \mathcal{P}_r, d \in \mathcal{R}^{\mathcal{D}}_{rp}\}$
$\mathcal{D}^{\mathcal{R}}_p$	subset of $\mathcal{D}^{\mathcal{R}}$ for mother and daughter routes pairs that are connected using port p

Parameters

C^{MH}_k	weekly time charter cost of a mother vessel of type k
C^{RM}_{rk}	sailing cost of mother route r using vessel type k, composed of port, bunker and handling costs
C^D_d	total operational cost of daughter route d, including both weekly time charter costs, sailing costs, port costs, cargo handling costs, and transshipment costs
U_k	capacity in TEUs of a mother vessel of type k
D_p	demand of cargo (number of containers) to deliver to feeder port p from the hub port
P_p	demand of cargo (number of containers) to pick up from feeder port p for transportation to the hub port
L^-_d	total number of containers to deliver to feeder ports visited by daughter route d
L^+_d	total number of containers to pick up from feeder ports visited by daughter route d
H_E	cargo handling rate at the hub port (in TEUs per hour)
H_p	cargo handling rate at feeder port p (in TEUs per hour)
S_r	sailing time of mother route r (in hours)
N	the maximum number of daughter vessels that can do the transshipment at any feeder port

Decision variables

x_{rk}	takes value 1 if mother route r is sailed with vessel of type k and 0 otherwise
y_{rk}	number of mother vessels of type k used on mother route r
q_r	maximum number of containers carried on mother route r including containers of daughter routes connected to r
t_{rk}	total time (sailing time and cargo handling time) of mother route r with vessel type k
z_{rd}	takes value 1 if daughter route d is selected and connected to an active mother route r and 0 otherwise

The mathematical model of the underlying problem is as follows:

$$(F2) : \min \sum_{r\in\mathcal{R}^{\mathcal{M}}} \sum_{k\in\mathcal{K}} (C_k^{MH} y_{rk} + C_{rk}^{RM}) + \sum_{(r,d)\in\mathcal{D}^{\mathcal{R}}} C_d^D z_{rd} \tag{3.1}$$

subject to:

$$\sum_{k \in \mathcal{K}} x_{rk} \leq 1, \qquad r \in \mathcal{R}^{\mathcal{M}} \tag{3.2}$$

$$\sum_{r \in \mathcal{R}_p^{\mathcal{M}}} \sum_{k \in \mathcal{K}} x_{rk} + \sum_{(r,d) \in \mathcal{D}_p^{\mathcal{R}}} z_{rd} \geq 1, \qquad p \in \mathcal{P} \tag{3.3}$$

$$N \sum_{k \in \mathcal{K}} x_{rk} \geq \sum_{(r,d) \in \mathcal{D}^{\mathcal{R}}} z_{rd}, \qquad r \in \mathcal{R}^{\mathcal{M}} \tag{3.4}$$

$$\begin{aligned} q_r \geq & \sum_{p' \in \mathcal{P}r} D_{p'} \sum_{k \in \mathcal{K}} x_{rk} + \sum_{(r,d) \in \mathcal{D}^{\mathcal{R}}} z_{rd} L_d^- z_{rd} \\ & + \sum_{p' \in P_{rp}^-} \Big((P_{p'} - D_{p'}) \sum_{k \in \mathcal{K}} x_{rk} + \sum_{(r,d) \in \mathcal{D}_{p'}^{\mathcal{R}}} (L_d^+ - L_d^-) z_{rd} \Big), \qquad r \in \mathcal{R}^{\mathcal{M}}, p \in \mathcal{P}_r \end{aligned} \tag{3.5}$$

$$\sum_{k \in \mathcal{K}} U_k x_{rk} \geq q_r, \qquad r \in \mathcal{R}^{\mathcal{M}} \tag{3.6}$$

$$\begin{aligned} t_{rk} \geq & S_r x_{rk} + \sum_{p \in \mathcal{P}_r} \Big(\frac{P_p + D_p}{H_E} + \frac{P_p + D_p}{H_p} \Big) x_{rk} \\ & + \sum_{p \in \mathcal{P}_r} \sum_{(r,d) \in \mathcal{D}^{\mathcal{R}p}} \Big(\frac{L_d^+ + L_d^-}{H_E} + \frac{L_d^+ + L_d^-}{H_P} \Big) z_{rd}, \qquad r \in \mathcal{R}^{\mathcal{M}}, k \in \mathcal{K} \end{aligned} \tag{3.7}$$

$$y_{rk} \geq \frac{t_{rk}}{168}, \qquad r \in \mathcal{R}^{\mathcal{M}}, k \in \mathcal{K} \tag{3.8}$$

$$x_{rk} \in \{0, 1\}, \qquad r \in \mathcal{R}^{\mathcal{M}}, k \in \mathcal{K} \tag{3.9}$$

$$y_{rk} \in \mathcal{N}, \qquad r \in \mathcal{R}^{\mathcal{M}}, k \in \mathcal{K} \tag{3.10}$$

$$y_{rd} \in \{0, 1\}, \qquad (r, d) \in \mathcal{D}^{\mathcal{R}} \tag{3.11}$$

$$q_r \geq 0, \qquad r \in \mathcal{R}^{\mathcal{M}} \tag{3.12}$$

$$t_{rk} \geq 0, \qquad r \in \mathcal{R}^{\mathcal{M}}, k \in \mathcal{K} \tag{3.13}$$

The objective function 3.1 aims to minimize the weekly operational costs. This cost calculation comprises two components. The first and second terms estimate the weekly operational expenses for the chosen mother and daughter routes. These weekly operational costs encapsulate the weekly TC rate for the mother or daughter vessels in use, port expenses, fuel charges, and costs related to cargo handling. Additionally, the second term incorporates the transshipment costs.

Constraint 3.2 ensures that each route is sailed by only one type of vessel. Constraint 3.3 requires each port to be visited either by a mother or a daughter route, while Constraint 3.4 specify that any selected daughter route must be linked to a mother route.

Constraint 3.5 calculates the maximum volume of containers (measured in TEUs) transported by a mother route. The first term on the right side determines the number of containers delivered to various feeder ports along the mother route, and the second term calculates the volume of containers delivered to daughter routes at transshipment points along the same route. These two quantities comprise the initial load of containers at the hub. The third term accounts for the containers picked up and delivered for the ports in the mother route and connected daughter routes. By considering these values, the correct maximum load of the mother routes can be computed. This information is then utilized in Constraint 3.6 to identify the necessary vessel size from which the vessel type can be derived.

Constraint 3.7 computes the total duration of routes. This includes sailing time and cargo handling time needed for delivering and picking up containers to and from the visited ports. The number of mother vessels on a mother route, as regulated by Constraint 3.8, is determined by the number of weeks required to sail the route. Lastly, Constraints 3.9 to 3.13 define the domain of the decision variables.

3.2 FNDP-OT with OD patterns

The Origin-Destination (OD) variant builds upon the problem description from Chapter 3.1, with a couple of notable distinctions. We now include the demand for cargo to be transported to and from the central hub port and between feeder ports. This means instead of considering just a set of ports with demand flowing to and from the hub, the problem also incorporates a set of OD pairs, where a specific volume of cargo needs to be transported from one feeder port to another.

In this model, cargo may be transshipped up to three times before reaching its final destination. For example, it could be moved at a transshipment port, at the hub, and then at another transshipment port. Additionally, demands between feeder ports are considered optional and can be rejected. If cargo is rejected, a penalty is incurred for each container not delivered (1000 $). Rejection is all-or-nothing - either all containers of the demand are handled, or all are rejected. We denote this problem as Feeder Network Design Problem with Origin-Destination (FNDP-OT with OD-patterns).

3.2.1 Speed Optimization

We included a speed optimization algorithm that calculates the ideal speed s for a vessel v. The reason behind this is to increase flexibility, regulations, asset utilization, and fuel efficiency. The speed of a vessel has a significant impact on its fuel consumption. In general, slower speeds result in less fuel use, which can dramatically decrease operating costs. Considering the volatility of fuel prices and the growing concern about environmental sustainability, improving fuel efficiency has become a key objective in the shipping industry.

Algorithm 1: Speed optimization algorithm

```
Function SpeedOpt(distance, peak_load, load_time, set_of_vessels V):
    min_cost ← $\infty$
    vessel ← $None$
    speed ← 0
    foreach $v \in$ V such that $v_{capacity} \geq$ peak_load do
        $t$ ← calculate sail time for the given distance at speed $v_{max}$
        weeks ← $\lceil (t + load\_time)/(24 \times 7) \rceil$
        if weeks > time_limit then
            continue
        end
        $t_{max}$ ← (weeks $\times 24 \times 7$) - load_time
        $x \leftarrow t/t_{max}$
        $s \leftarrow \max(v_{max} \times x, v_{min})$
        costs ← calculate route with regards to $v$ and $s$
        if costs < min_cost then
            min_cost ← costs
            vessel ← $v$
            speed ← $s$
        end
    end
    return if vessel = null then infeasible else (vessel, speed)
```

Algorithm 1 processes each vessel with a capacity equal to or greater than the peak load. For every individual vessel, the sail time is calculated at maximum speed. If the calculated time is feasible, it implies that the route itself is also feasible. The algorithm then determines the lowest possible speed required to complete the journey within the set time limit. Ultimately, it returns the selected vessel along with its optimal speed.

3.2.2 Mathematical Model

We also include a mathematical model by Msakni et al. (2020) for the FNDP-OT with OD-patterns to provide a clear mathematical representation of the problem after including OD-patterns and speed optimization. The different sets and parameters, as well as decision variables, are defined in the following:

Sets

$\mathcal{R}^{\mathcal{M}}$	set of mother routes.
$\mathcal{R}^{\mathcal{D}}$	set of daughter routes.
$\mathcal{R}$	set of all routes, $\mathcal{R} = \mathcal{R}^{\mathcal{M}} \cup \mathcal{R}^{\mathcal{D}}$.
$\mathcal{K}_r$	set of available ship sizes to sail route r.
$\mathcal{S}_k$	set of vessel speeds that can be sailed by a ship size $k, k \in \mathcal{K}$.
$\mathcal{R}^{\mathcal{M}}_p$	subset of mother routes that include a visit to port p.
$\mathcal{R}^{\mathcal{D}}_{\mathcal{P}}$	subset of daughter routes that do transshipment at port p.
$\mathcal{P}$	set of local ports.
$\mathcal{P}_r$	set of port visited by route r.
$\mathcal{D}^{\mathcal{R}}_j$	set of route pairs of mother and daughter routes that can do transshipment at port $j, (m, d) \in \mathcal{D}^{\mathcal{R}}_j$ if $m \in \mathcal{R}^{\mathcal{M}}, d \in \mathcal{R}^{\mathcal{D}}, j \in \mathcal{P}_m$, the transshipment port of d is j, and the ports visited by d are not visited by m (except for the transshipment port).
$\mathcal{D}^{\mathcal{R}}$	set of mother and daughter route pairs that can be connected.
$\mathcal{A}$	set of port pairs with demand $(a, b) \in \mathcal{A}, a, b, \in \mathcal{P}$.
$\mathcal{A}^{\mathcal{M}}$	set of mandatory demand, which is related to demand from or to the continental port.
$\mathcal{A}^{\mathcal{O}}$	set of optional demand, which is related to any demand between local ports.
$\bar{\mathcal{A}}_d$	subset of port pair demands (a, b) that is delivered by daughter route d $(a, b \in \mathcal{P}_d)$, where the cargo of delivering (a, b) goes through the transshipment port of d.

$\mathcal{E}$	set of all possible edges in the model, $(i, j) \in \mathcal{E}$, where $i, j \in \mathcal{P}$.
$\mathcal{E}_r$	set of edges of route r.

Parameters

C_k^{MH}	weekly time charter cost of a mother vessel of type k.
C_{rks}^{RM}	operation cost of mother route r sailing at speed s using vessel type k; this cost includes fuel cost and port calls.
C_d^D	total cost of daughter route d; the cost includes the weekly time charter, bunker, and transshipment costs.
U_k	capacity of vessel of type k (in FEU).
F_j	cargo handling rate at port j (in FEU per hour).
H_j	cargo handling cost at port j.
T_{rs}	sailing time of mother route r with speed s (in hours).
D_{ab}	the number of containers to transport from a to $b, (a, b) \in A$.
O_{ab}	Revenue from transporting one container of the optional demand $(a, b), (a, b) \in \mathcal{A}^{\mathcal{O}}$.
$\bar{O_{ab}}$	Penalty cost for not serving one container from the optional demand $(a, b), (a, b) \in \mathcal{A}^{\mathcal{O}}$.
N	the maximum number of daughter vessels that can do the transshipment at a port.
M	a big value.

Decision variables

x_{rks}	a binary variable that takes value 1 if vessel type k is used to sail route r with speed s, and 0 otherwise.
y_{rk}	a positive integer variable that indicates the number of mother vessels of type k used on mother route r.

$l_{ij}^{(ab)}$	number of containers of demand (a, b) traveling through edge (ij).
$z_j^{r(ab)}$	demand (a, b) that enters the system from port j using route r.
c_{rj}	transshipped cargo transported by mother route r using route j.
t_{rk}	total sailing time and cargo handling time for mother route r with vessel type k.
$u_{(r',r)}^{(ab)}$	a binary variable that takes value 1 if demand (a, b) is transshipped between routes r and r', and 0 otherwise.

$$
\begin{aligned}
(F1) : \min \sum_{m\in\mathcal{R}^{\mathcal{M}}} \sum_{k\in\mathcal{K}_m} \sum_{s\in\mathcal{S}_k} \left(C_k^{MH} y_{mk} + C_{mks}^{RM} x_{mks}\right) + \sum_{d\in\mathcal{R}^{\mathcal{D}}} \sum_{k\in\mathcal{K}_d} \sum_{s\in\mathcal{S}_k} C_d^D x_{dks} + \\
2\sum_{j\in\mathcal{P}} \sum_{(r,r')\in\mathcal{D}_j^{\mathcal{R}}} H_j u_{rr'}^{(ab)} + \sum_{(a,b)\in\mathcal{A}^O} \sum_{r\in\mathcal{R}} \left(D_{ab} - z_a^{r(ab)}\right)\bar{O}_{ab} - \sum_{(a,b)\in\mathcal{A}^O} \sum_{r\in\mathcal{R}} O_{ab} z_a^{r(ab)}
\end{aligned}
\tag{3.14}
$$

subject to:

$$\sum_{k\in\mathcal{K}_r}\sum_{s\in\mathcal{S}_k} x_{mks} \leq 1, \qquad r\in\mathcal{R} \tag{3.15}$$

$$N\sum_{k\in\mathcal{K}}\sum_{s\in\mathcal{S}_k} x_{mks} \geq \sum_{(r,d)\in\mathcal{D}^{\mathcal{R}}}\sum_{k\in\mathcal{K}_d}\sum_{s\in\mathcal{S}_k} x_{dks}, \qquad r\in\mathcal{R}^{\mathcal{M}} \tag{3.16}$$

$$\sum_{m\in\mathcal{R}_p^M}\sum_{k\in\mathcal{K}_m}\sum_{s\in\mathcal{S}_k} x_{mks} \leq 1, \qquad p\in\mathcal{P} \tag{3.17}$$

$$\sum_{m\in\mathcal{R}_p^D}\sum_{k\in\mathcal{K}_d}\sum_{s\in\mathcal{S}_k} x_{dks} \leq 1, \qquad p\in\mathcal{P} \tag{3.18}$$

$$\begin{aligned} &l_{ij}^{r(ab)} + z_j^{r(ab)} + \sum_{(r',r)\in\mathcal{R}_j^{\mathcal{D}}} u_{(r'r)}^{(ab)} = \\ &\quad l_{jk}^{r(ab)} + \sum_{(r',r)\in\mathcal{R}_j^{\mathcal{D}}}, \qquad r\in\mathcal{R},(a,b)\in\mathcal{A},(i,j)\in\mathcal{E}_r,(j,k)\in\mathcal{E}_r, j\neq b \end{aligned} \tag{3.19}$$

$$\sum_{r\in\mathcal{R},a\in\mathcal{P}_r} z_a^{r(ab)} = D_{ab}, \qquad (a,b)\in\mathcal{A}^M \tag{3.20}$$

$$\sum_{r\in\mathcal{R},a\in\mathcal{P}_r} z_a^{r(ab)} = D_{ab}, \qquad (a,b)\in\mathcal{A}^O \tag{3.21}$$

$$\sum_{(a,b)\in\mathcal{A}} l_{ij}^{r(ab)} \leq \sum_{k\in\mathcal{K}_r}\sum_{s\in\mathcal{S}_k} U_k x_{rks}, \qquad r\in\mathcal{R},(i,j)\in\mathcal{E}_r \tag{3.22}$$

$$\begin{aligned} t_{rk} \geq{}& T_{rs}\sum_{s\in\mathcal{S}_k} x_{rks} + M\Big(\sum_{s\in\mathcal{S}_k} x_{rks} - 1\Big) + \sum_{j\in\mathcal{P}_r}\Big(\sum_{\substack{(a,j)\in\mathcal{A},\\(i,j)\in\mathcal{E}_r}} l_{ij}^{r(aj)} + \\ &\sum_{\substack{(j,b)\in\mathcal{A},\\(j,k)\in\mathcal{E}_r}} l_{jk}^{r(jb)} + \sum_{(a,b)\in\mathcal{A}}\sum_{(r',r)\in\mathcal{R}_j^D}\big(u_{(r',r)}^{(ab)} + u_{rr'}^{(ab)}\big)\Big)/F_j, \qquad r\in\mathcal{R}^M, k\in\mathcal{K}_r \end{aligned} \tag{3.23}$$

$$q_{mk} \geq \frac{t_{mk}}{168}, \qquad m\in\mathcal{R}^M, k\in\mathcal{K}_m \tag{3.24}$$

$$x_{rks}\in\{0,1\}, \qquad r\in\mathcal{R}, k\in\mathcal{K}, s\in\mathcal{S}_k \tag{3.25}$$

$$y_{rk}\in\mathbb{N}, \qquad r\in\mathcal{R}^M, k\in\mathcal{K} \tag{3.26}$$

$$u_{rr'}^{(ab)}\in\{0,1\}, \qquad (r,r')\in\mathcal{R}_j^D \tag{3.27}$$

$$z_j^{r(ab)} \geq 0, \qquad r\in\mathcal{R}, j\in\mathcal{P}_r, (a,b)\in\mathcal{A} \tag{3.28}$$

$$l_{ij}^{r(ab)} \geq 0, \qquad r\in\mathcal{R}, (i,j)\in\mathcal{E}^r, (a,b)\in\mathcal{A} \tag{3.29}$$

$$t_{rk} \geq 0, \qquad r\in\mathcal{R}^M, k\in\mathcal{K}_r \tag{3.30}$$

The objective function 3.14 aims to minimize the weekly operational costs. The first term calculates the operational costs associated with the selected mother routes, while the second term computes the costs of the selected daughter routes. The third term determines the costs of transshipment between mother and daughter routes. Additionally, the fourth term computes the penalty for rejecting containers and the revenue generated from transporting selected containers.

Constraint 3.15 ensures that each selected route is assigned a specific vessel type and speed. Constraint 3.16 guarantees that a daughter route can only be selected if it can be connected to a mother route selected by the model. To maintain efficient operations, each port is served by either a mother or daughter route, as specified by Constraint 3.17 and Constraint 3.18.

Constraint 3.19 ensures the correct flow of containers, considering transshipment operations. Constraint 3.20 indicates that a mandatory demand (a, b) must enter the system through port a. Similarly, Constraint 3.21 states that an optional demand (a, b) may enter the system through port a.

Constraint 3.22 ensures that the load transported over an edge does not exceed the capacity of the selected vessel type for the corresponding route. Constraint 3.23 computes the total duration of routes, taking into account both sailing time and cargo handling time required to deliver and pick up containers at the visited ports. Constraint 3.24 specifies that for each mother route, the number of vessels deployed on the route must be greater than or equal to the number of weeks required to sail the entire route. Finally, Constraints 3.25 to 3.29 define the domain of the decision variables, ensuring that they satisfy specific ranges or conditions.

3.3 Solution Representation

Solution representation is how we represent our solution to a given problem. When creating the solution representation, one has to think about scalability. We have two similar but different types of problems we want to solve, where one is an extended version of the other. Therefore we utilize a variant of the solution representation proposed by Bergmann et al. (2023). The *solution representation* is divided into three main components for the FNDP-OT, and we include an optional array for the solution representation of FNDP-OT with OD-patterns to handle the OD.

Part 1 represents the sequence of ports visited by one or more mother vessels. The continental port (hub) is excluded from this representation since all mother routes start and end at the continental port. If there are multiple mother routes, they are separated using the mother token (-2).

Part 2 represents the remaining portion of the main vector and depicts the sequence of ports visited by daughter vessels, if applicable. Daughter routes are separated using the daughter token (0). In the case of butterfly routes, the butterfly token (-1) is used to indicate the separation between the first and second loop within a butterfly structure. The butterfly token (-1) marks the point in the port visit sequence where the daughter route returns to the transshipment port before continuing.

Transshipment vector represents the transshipment port for each daughter route, where the index is the daughter route and the value is its transshipment port (origin port). As seen in Figure 3.3, the first value is 2, which indicates that the origin port of *daughter route* 1 is port 2. I.e., the first daughter route is in fact $[2, 1, 2, 4, 2]$ and the second is then $[6, 3, 7, 8, 6]$.

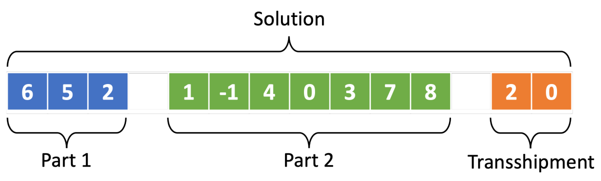


Figure 3.3: FNDP-OT

The **optional** field (Figure 3.4) is added for the Feeder Network Design Problem (FNDP) with OD-patterns. This field is an array of 1 and 0 representing boolean values for whether we choose to handle the "Origin-Destination" demands. The length of the optional array is restricted to the number of Origin-Destination (OD) demands.

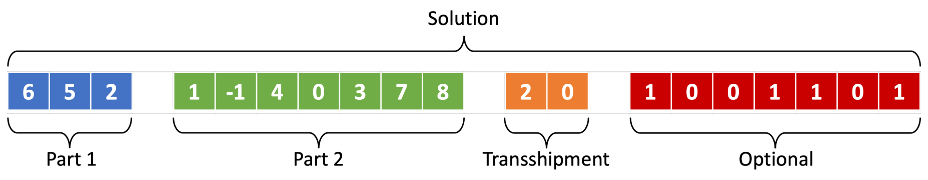


Figure 3.4: FNDP-OT Origin-Demand

3.4 Vessel Data

We use the same vessel data as Brouer et al. (2014) with the addition of a fifth larger mother vessel. Figure 3.7 provides an overview of the available vessels for both FNDP-OT and FNDP-OT with OD-patterns.

Table 3.7: Detailed overview of the different vessels available

Vessel	Capacity (TEU)	TC-rate (daily)	min speed	max speed	design speed	design cons	idle cons
Fleet A: Data imported from LINER-LIB							
Feeder 450	450	5 000	10	14	12.0	18.8	2.4
Feeder 800	800	8 000	10	17	14.0	23.7	2.5
Panamax 1200	1 200	11 200	12	19	18.0	52.5	4.0
Panamax 2400	2 400	21 000	12	22	16.0	57.4	5.3
Post panamax	4 200	35 000	12	23	16.5	82.2	7.4
Sumper panamax	7 500	55 000	12	22	17.0	126.9	10.0
Fleet B: Estimated data for smaller vessels							
Small feeder 100	100	1 300	8	13	12.0	10.0	1.9
Small feeder 200	200	2 500	9	13	12.0	12.0	2.1
Small feeder 300	300	3 600	9	14	12.0	16.0	2.2

Table 3.7 provides an overview of the different vessel types. This table encapsulates a range of information for each vessel class, including capacity, Time Charter (TC) rate, minimum speed, maximum speed, design speed, fuel consumption at design speed, and fuel consumption when idle. The table presents data on two distinct fleets. Fleet A, comprising larger vessels, was introduced by Brouer et al. (2014) (LINER-LIB), while Fleet B includes smaller feeder vessels. If we ignore speed optimization, the vessel will travel at design speed.

We utilize the same function for computing fuel consumption as proposed by Brouer et al. (2014). Given a specific vessel and a desired speed, the fuel consumption is calculated using the following formula:

$$f_{fuel}(v, s) = (\frac{s}{v_{design}})^3 v_{cons} \tag{3.31}$$

In this equation, v represents the vessel, s stands for the desired speed, v_{design} signifies the design speed of the vessel, and v_{cons} is the fuel consumption at the design speed. This calculation provides an efficient method for estimating the fuel consumption of a vessel at any given speed.

3.5 Instances

In our study, we utilized problem instances from multiple sources. Firstly, we employed the instances introduced by Msakni et al. (2020), which consisted of two sets. The first set included ports along the Norwegian coastline, while the second set is adapted instances from LINER-LIB Brouer et al. (2014). Both sets of instances were constructed using realistic data.

Table 3.8 provides an overview of the instances introduced by Msakni et al. (2020). The table includes information such as the number of feeder ports in each instance, a brief description, and the total number of demands measured in TEUs. It is worth noting that the instances are categorized into classes A, B, and C, where each class represents a smaller set of instances. For these classes, the reported numbers of ports and demands in Table 3.8 represent ranges rather than exact values.

Table 3.8: Detailed information about instance A, B, C and adapted LINER-LIB.

Category	#Ports	Description	Total demand (TEUs)
		Instances introduced by Msakni et al. (2020)	
Class A	10 - 11	Rotterdam - Norwegian Coastline	1070 - 1579
Class B	15	Rotterdam - Norwegian Coastline	1074 - 1725
Class C	20 - 22	Rotterdam - Norwegian Coastline	1850 - 4126
		Adapted LINER-LIB instances	
Baltic	12	Bremerhaven - Baltic sea	4 904
WAF	20	Algericas - West Africa	8 489
Medit 1	33	Algericas - Mediterranean - Canaris	2 705
Medit 2	32	Gioia Tauro - Mediterranean - Canaris	599

Chapter 4

Adaptive Heuristic

Adaptive Heuristic (AH) was first introduced in 2023, by the works of Bergmann et al., and showed remarkable results when applied to the FNDP-OT and FNDP-OT with OD-patterns.

4.1 An Overview of Adaptive Heuristic

The metaheuristic AH follow the same structure as ALNS. However, AH enhances these pre-existing heuristics by introducing an escape algorithm, a unique feature that facilitates its evasion from local optima. Besides its unique mechanism, AH has proven to be highly efficient, demonstrating impressive computational time results, further emphasizing its practical use.

While this thesis implements the framework in Python, not Kotlin, as in the original context, it is important to mention that the computational time may increase as a result. However, this change in programming language will not influence the outcome, ensuring the validity and comparability of the results.

Algorithm 3: Adaptive Heuristic

```
Function $AH$:
    Input: a set of heuristics $\mathbb{H}$
    Generate an initial solution $s$
    $s_{best} \leftarrow s$
    $i \leftarrow 0$
    repeat
        $s' = s$
        select $h \in \mathbb{H}$ based on selection parameters
        apply heuristic $h$ to $s'$
        if $f(s') < f(s_{best})$ then
            $s_{best} = s'$
            $i \leftarrow 0$
        end
        if $accept\ (s', s)$ then
            $s = s'$
            algorithm
        end
        update selection parameters and $i \leftarrow i + 1$
        if $Escape\_Condition(i, m)$ then
            $Escape\_Algorithm(s, s_{best})$
            $i \leftarrow 0$
        end
    until stop-criterion met
```

The Adaptive Heuristic follows the same pattern as ALNS framework. First, it generates an initial solution and updates *best_found_solution* accordingly. After doing so, it iterates over n number of iterations, selects a heuristic, applies it to the incumbent, and receives a new solution. Based on this new solution, the heuristic receives a score based on the improvement of the new solution compared to the *incumbent* the *best_found_solution.* If the solution has not improved in the last i steps, we utilize the *escape_algorithm* (section 4.2) to escape the local optimum.

4.1.1 Reward system

The reward system serves as a fundamental element within metaheuristics, guiding the algorithm in prioritizing strategies. This adaptability stands as a primary strength of metaheuristics. By leveraging insights from previous experiences, the algorithm refines its ability to recognize and employ the most promising heuristics. Consequently, this enables more efficient problem-solving processes and facilitates the generation of improved solutions. This adaptability proves especially valuable when tackling intricate combinatorial optimization problems characterized by vast and intricate solution spaces, where identifying the optimal solution is not readily apparent.

$$\sigma_h = \begin{cases} \sigma_1 & \text{if new solution beats best found solution} \\ \sigma_2 & \text{if acceptance criteria is met} \\ \sigma_3 & \text{if new solution is not previously seen} \\ \sigma_4 & \text{otherwise} \end{cases} \tag{4.1}$$

Bergmann et al. (2023) utilized the following reward system him is research: $\sigma_1 = 10, \sigma_2 = 7, \sigma_3 = 4, \sigma_4 = 0$. Which rewards exploration, meaning that a heuristic is rewarded when generating an unseen solution (exploration of the neighborhood) even though it has a worse objective value.

4.1.2 Acceptance criterion

The acceptance criterion used in Adaptive Heuristic is known in the literature as the *Metropolis acceptance criterion.* The probability function can be written as 4.2:

$$P(s' \mid s) = \begin{cases} 1 & \text{if } f(s') < f(s) \\ e^{-\frac{f(s)-f(s')}{T}} & \text{otherwise} \end{cases} \tag{4.2}$$

where s is the incumbent and s' is the new solution generated. $f(s)$ and $f(s')$ are the objective value of the incumbent and new solution, respectively. T is a parameter called the *temperature*, which is gradually reduced during the course of the algorithm. All better solutions are accepted. If the new solution is worse than the incumbent, it has a probability of $e^{-\frac{f(s)-f(s')}{T}}$ to be accepted, unless it is already seen, then it is declined. This probability function is designed to allow occasional acceptance of worse solutions, helping to avoid getting stuck in local optima. The decreasing temperature schedule reduces the chance of accepting worse solutions as the search progresses, encouraging convergence to an optimal solution.

Cooling Schedule

Besides this, Bergmann et al. utilized the cooling schedule suggested by Crama and Schyns (2003). The cooling schedule is defined as $T_i = \alpha T_{i-1}$, where i is the iteration and α is the cooling factor. The initial temperature, T_0 is calculated by:

$$T_0 = \frac{-f^T_{avg}}{ln(p)} \tag{4.3}$$

To tune this, AH runs n iterations with a fixed acceptance probability p and calculates the average f^T_{avg} of all $|f(s) - f(s')|$ for all solutions that are worse but accepted, after this, we can use the initial temperature T_0, final temperature T_f and the number of iterations to calculate the cooling factor α:

$$\alpha = \left(\frac{T_f}{T_0}\right)^{\frac{1}{n}} \tag{4.4}$$

4.2 Escape algorithm

The *escape algorithm* used in AH is applied if the solution has not improved during the n last iterations. This means that the search is stuck in a local optimum and needs to be *moved* to another place in the solution space. Bergmann et al. suggested the escape algorithm outlined in Algorithm 4.

Algorithm 4: Escape Algorithm for AH

```
Function Escape_Algorithm(s, $s_{best}$):
    reject each demand with a probability $p_{esc}$
    repeat
        choose a random heuristic $h \in \mathbb{H}$
        $s \leftarrow h(s)$
        if $f(s) < f(s_{best})$ then
            $s_{best} \leftarrow s$
        end
    until stop-criterion met
    return s, $s_{best}$
```

The first thing the escape function does is reject OD demands with a probability p. This means that each demand already handled has an equal probability of being ignored. After this, it randomly selects heuristics and performs changes to the solution. This solution is then the base for the continuing search.

4.3 Selecting heuristics and weight adjustment

At the outset, every heuristic, denoted as $h \in \mathbb{H}$, is assigned an initial weight w_h. The probability of choosing heuristic h in the set of heuristics $\mathbb{H}$ is given by $\frac{w_h}{\sum_{g \in \mathbb{H}} w_g}$. Where w_h is the weight of the heuristic h.

The selected heuristic is rewarded based on the degree of improvement it achieves. The scoring system consists of three tiers: **(i)** *high score* is awarded if the heuristic discovers a new best solution. **(ii)** *medium score* is given if the heuristic finds a solution better than the current one. **(iii)** *low score* is assigned if the heuristic uncovers a solution that has not been encountered before.

$$w_{h,s} = w_{h,(s-1)}r + (1-r)\frac{\pi_h}{\lambda_h} \tag{4.5}$$

At the end of each segment, the weights are updated by considering the number of times each heuristic has been used and the cumulative scores it has obtained (Equation 4.5). This update allows the system to adapt and allocate more weight to heuristics that consistently perform well. After updating the weights, the scores and use counter are reset.

4.4 Heuristics

This section describes various heuristics utilized by the AH suggested by Bergmann et al. (2023). Each heuristic functions uniquely and addresses a specific component of the solution representation. By employing a combination of these heuristics, our approach achieves a balanced blend of intensification on promising areas and diversification of diverse solutions throughout the search process.

4.4.1 Random swap

The heuristic first checks if the size of *Part 1* (mother routes) is less than 2. If there are fewer than 2 ports in the *Part 1*, the function returns *infeasible*, indicating that the heuristic cannot be performed. If this is not the case, the heuristic proceeds with the swapping operation. It selects two random indices and swaps the values. We return the new solution if this results in a new *feasible* solution. If not, it returns *infeasible*.

4.4.2 Reinsert in mother

The heuristic first checks if the size of *Part 1* (mother routes) is less than 2. If there are fewer than 2 ports in the *Part 1*, the function returns *infeasible*, indicating that the heuristic cannot be performed. If this is not the case, the heuristic gathers all indices of port values in *Part 1* and removes a random port (denoted x). It then tries to insert x in all possible positions in *Part 1*, including a separate mother route for port x. The transshipment vector is updated accordingly. If this results in any new *feasible* solutions, we return the solution with the lowest objective value. If not, it returns *infeasible*.

4.4.3 Reinsert in daughter

This heuristic randomly moves a port from *Part 2* (daughter routes) and tries to reinsert it. All possible insertions are tested to keep the best option, including having a new daughter route. If the selected port is the only port in the original daughter route, this route is removed, and its corresponding transshipment entry is also removed. If the original daughter route is a butterfly with only the selected port in one of its loops, it is converted to a simple daughter route.

4.4.4 Mother route to daughter route

The heuristic first checks if the size of *Part 1* (mother routes) is less than 2. If there are fewer than 2 ports in the *Part 1*, the function returns *infeasible*, indicating that the heuristic cannot be performed. If this is not the case, the heuristic gathers all indices of port values in *Part 1* and removes a random one (denoted x). It then tries to insert x in all possible positions in *Part 2*, including a separate daughter route for port x. The transshipment vector is updated accordingly. If this results in any new *feasible* solutions, we return the solution with the lowest objective value. If not, it returns *infeasible*.

4.4.5 Daughter route to mother route

The heuristic gathers all indices of port values in *Part 2* (daughter routes) and removes a random one (denoted x). It then tries to insert x in all possible positions in *Part 1*, including a separate daughter route for port x. The transshipment vector is updated accordingly. If this results in any new *feasible* solutions, we return the solution with the lowest objective value. If not, it returns *infeasible*.

4.4.6 Butterfly route to simple route

The heuristic iterates through *Part 2* (daughter routes) and collects all daughter routes that contain the *butterfly token*. If there are no *butterfly token* in *Part 2*, the function returns *infeasible*, indicating that the heuristic cannot be performed. If this is not the case, the heuristic selects one daughter route containing a butterfly route and removes the *butterfly token*, making it a *simple route*. We return the solution if this results in a new *feasible* solution. If not, it returns *infeasible*.

4.4.7 Simple route to butterfly route

The heuristic iterates through *Part 2* (daughter routes) and collects all daughter routes that do not contain the *butterfly token*. If there are no simple routes in *Part 2*, the function returns *infeasible*, indicating that the heuristic cannot be performed. If this is not the case, the heuristic selects one daughter route and tries to insert *butterfly token* in all possible positions in that daughter route, making it a *butterfly route*. We return the solution if this results in a new *feasible* solution. If not, it returns *infeasible*.

4.4.8 Optimize transshipment vector

The heuristic selects one random value in the transshipment vector. We then iterate through *Part 1* (mother routes) and calculate the cost for each port as a feeder port. We return the best-found solution if this results in a new *feasible* solution. If not, it returns *infeasible*.

4.4.9 Enable demand

The heuristic iterates through the *Optional vector* and randomly selects one *False (0)* value. It changes the value to *True (1)* and then returns the new solution if *feasible*. If not, it selects another random *False (0)* value and tries to reinsert the feeder port and select another vessel.

4.4.10 Reject demand

The heuristic iterates through the *Optional vector* and randomly selects one *True (1)* value. It changes the value to *False (0)* and then returns the new solution if *feasible*. If not, it returns infeasible.

Chapter 5

DRLH

In this chapter, we introduce the Deep Reinforcement Learning Hyperheuristic (DRLH). DRLH is an innovative approach that combines Deep Reinforcement Learning with ALNS to solve COPs. DRLH showed great promise when applied to Capacitated Vehicle Routing Problem, Parallel Job Scheduling Problem, Pickup and Delivery problem, and Pickup and Delivery Problem with Time Windows (Kallestad et al. (2023)), which are all challenging combinatorial optimization problems in various domains.

5.1 Deep Reinforcement Learning Hyperheuristic

DRLH, which stands for Deep Reinforcement Learning Hyperheuristic, is a hyperheuristic first introduced by Kallestad et al. (2023). The framework utilizes an RL agent for the selection of heuristics. This process enhances the ALNS framework proposed by Ropke and Pisinger (2006) by integrating the decision-making capability of the RL agent, allowing it to select the most suitable heuristic to apply to the solution at micro-level for each iteration. The pseudocode is shown in Algorithm 5.

Algorithm 5: DRLH

```
Function Deep Reinforcement Learning Hyperheuristic:
    Generate an initial solution $s$ with objective function $f(s)$
    $H \leftarrow set\_of\_heuristics$
    $s_{best} \leftarrow s$
    $f(s_{best}) \leftarrow f(s)$
    repeat
        $s' = s$
        select $h \in \mathbb{H}$ based on policy $\pi(h|s,\theta)$
        apply heuristic $h$ to $s'$
        if $f(s') < f(s_{best})$ then
            $s_{best} = s'$
        else
            if accept $(s', s)$ then
                $s \leftarrow s'$
            end
        end
    until stop-criterion met
    return $s_{best}$
```

DRLH takes in a set of heuristics and generates an initial solution. After doing so, it iterates for a number of iterations, applies a heuristic on the incumbent, and receives a reward depending on the level of improvement.

5.2 Heuristics

While the original work proposed by Kallestad et al. (2023) introduced an innovative methodology for generating heuristics by creating a diverse set of heuristics, denoted as H, there is still room for improvement. This set, H, was assembled by evaluating all possible permutations of the available *removal* and *insertion* operators, thereby forming a comprehensive collection of heuristic strategies for solving COPs. However, this might cause problems when switching between different subcategories of COPs. For the FNDP-OT and FNDP-OT with OD-patterns, there is a need for problem-specific operators; therefore, a potential enhancement might involve incorporating complete heuristics directly into the set of heuristics H from the start based on the specific problem, rather than relying on the generation from permutations of operators alone. This direct inclusion might save computational time and resources, enhancing the efficiency of the system. Leveraging complete heuristics will also present an opportunity for incorporating expert knowledge or high-performing heuristics from prior runs or related problems, making the approach more adaptive and versatile when comparing the performance with the AH.

Table 5.1: The different *heuristics* available to the DRLH for the FNDP and FNDP-OT.

Name	**Description**
Random_swap	Swaps two random ports in part 1
Reinsert_mother	Removes a port from part 1 and reinserts it into part 1
Reinsert_daughter	Removes a port from part 2 and reinserts it into part 2
Mother_to_daughter	Removes a port from part 1 and reinserts it into part 2
Daughter_to_mother	Removes a port from part 2 and reinserts it into part 1
Butterfly_to_simple	Randomly select a butterfly route and converts it into a simple route
Simple_to_butterfly	Randomly select a simple route and converts it into a butterfly route
Optimize_transshipment	Selects a random value from the transshipment vector and finds the optimal value for it
Enable_demand	Enables a randomly selected demand
Reject_demand	Rejects a randomly selected demand

As shown, the heuristic pool utilized (Table 5.1) is not the same as proposed by Kallestad et al. (2023), but rather the one proposed for the AH by Bergmann et al. described in chapter 4.4. Even though one of the many advantages of a hyperheuristic is to create or, simply put, combine removal and insertion operators, this gives us a clearer view of how to measure and evaluate performance when comparing to AH.

5.3 Acceptance Criteria and Stopping Conditions

DRLH uses the same acceptance criteria and stopping condition described in section 4.1.2. Where all better solutions are accepted, and worse solutions are accepted with a probability p unless it is previously seen.

5.4 Deep Reinforcement Learning Agent

The DRLH utilizes an *Actor-Critic* architecture with Proximal Policy Optimization (PPO) algorithm to update its policy (π_θ). The reason behind doing so is that we get the best of both *value-based* and *policy-based* methods. The *Actor* selects actions based on its current policy, while the *Critic* estimates the value function. PPO iteratively updates the policy by optimizing a surrogate objective while ensuring stability through the use of a policy clipping mechanism (Eq: 2.1). The implementations of the actor and critic networks enable the agent to learn complex mappings from *states* to *actions* and value estimates, facilitating effective decision-making in complex environments such as FNDP-OT and FNDP-OT with OD-patterns or other COPs.

5.4.1 State Representation

The state representation for FNDP-OT and FNDP-OT with OD-patterns builds upon the original state representation introduced by Kallestad et al. (2023), by adding three additional values. These include *sorted_seen_solution*, *sorted_accepted_solution*, and *sorted_new_best_solution*, each with a range from $[0, 1]$. For instance, *sorted_seen_solution* evaluates the quality of a new solution by comparing its rank among all previously seen solutions. Similar evaluations are conducted for *sorted_accepted_solution* and *sorted_new_best_solution*, providing comprehensive insight into the value of a given solution within the broader search context. It would be beneficial, in this scenario, to add more fields, such as *nr_daughter_routes*, *nr_mother_routes* or *opt_demands_handled.* However, we decided not to do this to keep it as a *general* framework.

Table 5.2: State representation for FNDP-OT and FNDP-OT with OD-patterns patterns

Name	**Definition**
improvement	The difference of cost between previous solution and current solution.
dist_to_best	The difference of cost between current solution and best-found solution.
inc_obj	The cost of the current solution.
best_obj	The cost of the best-found solution.
temperature	The current temperature.
cooling_rate	The cooling schedule (α).
no_improvement	The number of iterations since the last improvement.
index_step	The iteration number.
was_changed	1 if the solution was changed, 0 otherwise.
unseen	1 if the solution has not previously been seen in the search, 0 otherwise.
sorted_seen_solution	Value $[0, 1]$ showing the quality rank of the current solution among all seen solutions.
sorted_accepted_solution	Value $[0, 1]$ showing the quality rank of the current solution among all accepted solutions.
sorted_new_best_solution	Value $[0, 1]$ showing the quality rank of the current solution among all best solutions.
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.

5.4.2 Action

The number of actions is equal to the size of the heuristic pool H. In our case, we use the same heuristic pool as the AH (Table 5.1). At each *time step* (t) of the episode (search), the agent chooses a heuristic $h \in H$ and applies it to the solution giving us a new solution, feasible or not. Therefore we can define the policy function π as:

$$\pi(h|s, \theta) = Pr\{A_t = h|S_t = s, \theta_t = \theta\} \tag{5.1}$$

5.4.3 Reward function

The original DRLH framework by Kallestad et al. (2023) is adaptable to different reward functions such as:

$$R_t^{ALNS} = \begin{cases} 10 & \text{if } f(s') < f(s_{best}) \\ 7 & \text{if } f(s') < f(s) \\ 4 & \text{if } accept(s', s) \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

$$R_t^{5310} = \begin{cases} 5 & \text{if } f(s') < f(s_{best}) \\ 3 & \text{if } f(s') < f(s) \\ 1 & \text{if } accept(s', s) \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

$$R_t^{PM} = \begin{cases} 1 & \text{if } f(s') < f(s_{best}) \\ -1 & \text{otherwise} \end{cases} \tag{5.4}$$

where each serves a purpose. A good reward function is crucial in driving the agent toward desirable behavior and away from undesirable ones. It should accurately represent the task's objective and incentivize the agent to achieve the intended goal efficiently. Poorly designed reward functions may lead to unintended consequences, where the agent finds ways to "cheat" by achieving high rewards without actually fulfilling the intended task. Therefore we had to experiment with different reward functions to avoid reward hacking Skalse et al. (2022).

5.4.4 Training loop

The training loop for the DRLH agent is illustrated in Algorithm 6. It initializes a random policy, and for each episode, it resets the problem and executes n heuristic changes to the incumbent. After each episode, the policy parameters are adjusted according to PPO.

Algorithm 6: DRLH training loop

```
Result:
    Initialize a random policy $\pi_\theta$
    for $e \leftarrow 1$ to episode do
        $s_0 \leftarrow$ initial state $S_{t=0}$
        for $t \leftarrow 1$ to steps do
            Choose action $a \in A_t$ according to $\pi(a|s, \theta)$
            Perform $a$ and receive $R_t = v$ and $s \in S_{t+1}$
        end
        Update policy parameters $\theta$ according to PPO (Schulman et al. (2017))
    end
```

5.5 Solution Representation and Initial Solution

In both FNDP variants, we adopt the same solution representation detailed in sub-chapter 3.3. It is vital to acknowledge that each mother ship embarks from the Hub node, despite the hub node's absence in the state representation, to avoid redundancy.

We initialize the solution by assigning a mother vessel to each port, which implies that the number of mother vessels equals the number of ports. The vessel selected is the smallest one capable of accommodating the two-way cargo. This approach provides a solid starting point for problem-solving. For FNDP-OT with OD-patterns, we initialize the optional field with no OD-patterns being accepted.

Chapter 6

Experimental Setup

We have used the instances introduced by Msakni et al. (2020) to compare AH and DRLH, a set of adapted instances from LINER-LIB (Brouer et al. (2014)) based on the work by Msakni et al. (2020). For the OD version, randomly generated demands between the feeder ports have been added.

6.1 Experimental Environment

The computational experiments in this thesis were run on a 64-bit Windows 11 operating system with an AMD Ryzen 7 5800X 8-Core Processor (3.80 GHz) and 32GB DDR5 RAM.

6.1.1 Training of DRLH

The training of Deep Reinforcement Learning Hyperheuristic (DRLH) was performed on a remote server with an AMD EPYC 7742 64-Core Processor with a clock speed of 3.388 GHz, 256 CPUs, 64 cores per socket, and 8 NUMA nodes CPU and an NVIDIA A100, 80 GB Tensor Core GPU. The reason behind this was due to long training times, and the computational power needed exceeded local options. For instance, to train 500 iterations of FNDP-OT or FNDP-OT with OD-patterns with size 150 on took approximately 120 hours

6.2 Baseline Models

The baseline framework is Adaptive Heuristic. The reason behind this is that it outperformed ALNS and is, therefore, a solid baseline for the comparison of DRLH.

6.2.1 Adaptive Heuristic (AH)

As our approach is improving on the Adaptive Heuristic (AH) algorithm, this method is chosen as a baseline for performance comparison. This framework measures the performance of each heuristic using a scoring function identical to the one in DRLH in a segment. At the end of each segment, the weights of each heuristic are updated based on the performance in the segment.

6.3 Hyperparameter and Parameter Selection

Hyperparameters play a crucial role in machine learning models, including Deep Reinforcement Learning Hyperheuristic framework. These parameters are set before the training process and can significantly impact the model's performance, training speed, and stability.

6.3.1 Hyperparameters for DRLH

While training the DRLH agent on FNDP-OT and FNDP-OT with OD-patterns-patterns, we decided on hyperparameters found in table 6.1

Table 6.1: The hyperparameters used during training for the Agent of DRLH

Hyperparameter	**Value**
reward_func	R_t^{5310}
max_epochs	5000
learning_rate (α)	$1e-5$
batch_size	64
first_hidden_layer (size)	256
second_hidden_layer (size)	256
discount_factor (γ)	0.95

After conducting thorough testing, we carefully selected these hyperparameters to optimize our model's performance. We explored various values within a predefined range to identify the most suitable hyperparameters.

6.3.2 Parameters for AH

The process of selecting parameters for the AH was straightforward. The parameters for the original framework were optimized for a run of 20,000 iterations. We scaled these parameters down for a more manageable count of 1,000 iterations, adjusting the remaining parameters accordingly Table 6.2. Additionally, we conducted an exploration of various parameter combinations to ascertain the best comparative analysis.

Table 6.2: The parameters used for AH

Hyperparameter	**Value**
nr_its	1 000
segment_size	200
escape_its	50
tune_its	200
historic_weight	0.8
penalty	1 000
high_score	10
medium_score	7
low_score	4

These chosen values aim to balance optimal performance while preserving both intensification and diversification in the search space.

6.4 Dataset Generation

We generate a distinct training set of 5000 instances for all the instance sizes and a designated test set of 100. These datasets are generated completely randomly. They have no predefined or set values, all values for both distance, cargo, and OD patterns scale, within a given predefined range. These are generated to train and evaluate the DRLH agents performance.

Chapter 7

Results

7.1 Results of FNDP-OT

The results presented in this section are strictly bound to Feeder Network Design Problem with Optional Transsshipment, meaning that there is no *speed optimization* for the vessels, nor do they consider *OD-patterns* within the instances. Consequently, this leads to an augmented higher initial cost and reduced heuristic selection. DRLH and AH was applied to 12 instances with a number of ports ranging from 10 to 22. The initial costs ranged from 1,855 044.00 to 6,068,586.00. Each algorithm was executed 10 times on each instance to ensure comprehensive and robust results.

7.1.1 ABC instances

Table 7.1 displays the DRLH effectiveness at reducing cost in the ABC instances. DRLH reduced these costs significantly. The best-found objective ranges from 201,886.00 to 555,724.00, while the average objective ranges from 217,419.80 to 618,605.40 across all 12 instances. This indicates substantial improvement from the initial objective to the best-found objective on all instances, estimating an average improvement of 89.57%. The relatively high improvement rates across instances and the reasonable computational times highlight the potential of DRLH as a valuable tool for optimizing feeder networks.

Table 7.1: Detailed results on DRLH applied to the ABC instances

Instance	#Ports	Initial	Avg	Best	Improvement	Time
A1	10	1 855 044.00	217 419.80	201 886.00	89.12%	2.84
A2	10	2 284 739.00	238 120.10	230 797.00	89.90%	3.52
A3	11	2 259 021.00	228 315.70	226 254.00	89.98%	3.19
A4	11	2 555 751.00	284 646.50	258 016.00	89.90%	3.13
B1	15	3 010 390.00	441 917.70	350 340.00	88.36%	7.59
B2	15	2 624 494.00	419 610.00	299 323.00	88.60%	6.60
B3	15	2 337 815.00	477 891.80	313 290.00	86.60%	6.82
B4	15	2 907 184.00	372 691.80	275 908.00	90.51%	8.06
C1	20	3 611 139.00	528 345.30	433 627.50	87.99%	17.21
C2	22	4 166 586.00	412 932.30	346 586.00	91.68%	20.39
C3	22	4 851 586.00	466 047.90	420 590.00	91.33%	16.99
C4	22	6 068 586.00	618 605.40	555 724.00	90.84%	20.39
Average					89.57%	9.53

7.1.2 Performance comparison between DRLH and AH

To clarify, the headers on Table 7.2, are defined as follows:

- **Instance**: The name of the instance
- **Ports**: The number of ports (including HUB)
- **AH**: This records the best objective found over 10 runs using the Adaptive Heuristic (AH) framework
- **DRLH**: This displays the best objective found over 10 runs using the Deep Reinforcement Learning Hyperheuristic (DRLH) framework
- **Δ Best**: The difference between the best objectives found by AH and DRLH, calculated as AH best - DRLH best
- **Δ Avg**: The difference between the average objectives achieved by AH and DRLH, calculated as AH average - DRLH average

Table 7.2 demonstrates that, with the exception of one instance (B3), the DRLH consistently outperforms the AH algorithm on average over 10 runs, often by a significant margin. The improvements range from 49,153.60 (for instance B2) to as high as 1,657,665.62 (for instance C4). However, for instance B3, the AH algorithm performed better, with a marginal gain of 42,445.80. Despite the fact that the AH algorithm surpasses the best-found solution of DRLH in 4 out of 12 instances, DRLH is deemed more reliable due to its consistent results. These results provide strong evidence that the DRLH generally performs better than the AH in reducing cost and optimizing routes on these instances.

Table 7.2: Comparison of results on the ABC instances

Instance	Ports	AH	DRLH	Δ Best	Δ Avg
A1	10.0	192 558.0	201 886.0	-9 328.0	146 141.0
A2	10.0	233 361.0	230 797.0	2 564.0	109 664.5
A3	11.0	226 413.0	226 254.0	159.0	357 043.9
A4	11.0	287 296.0	258 016.0	29 280.0	158 366.0
B1	15.0	360 245.0	350 340.0	9 905.0	476 431.5
B2	15.0	282 332.0	299 323.0	-16 991.0	49 153.6
B3	15.0	251 169.0	313 290.0	-62 121.0	-42 445.8
B4	15.0	304 800.0	275 908.0	28 892.0	432 589.8
C1	20.0	424 755.0	433 627.5	-8 872.5	460 236.2
C2	22.0	474 312.0	346 586.0	127 726.0	856 176.7
C3	22.0	671 284.8	420 590.0	250 694.8	965 950.3
C4	22.0	753 171.5	555 724.0	197 447.5	1 657 665.6
Average		371 808.1	326 028.5	45 779.6	468 914.4

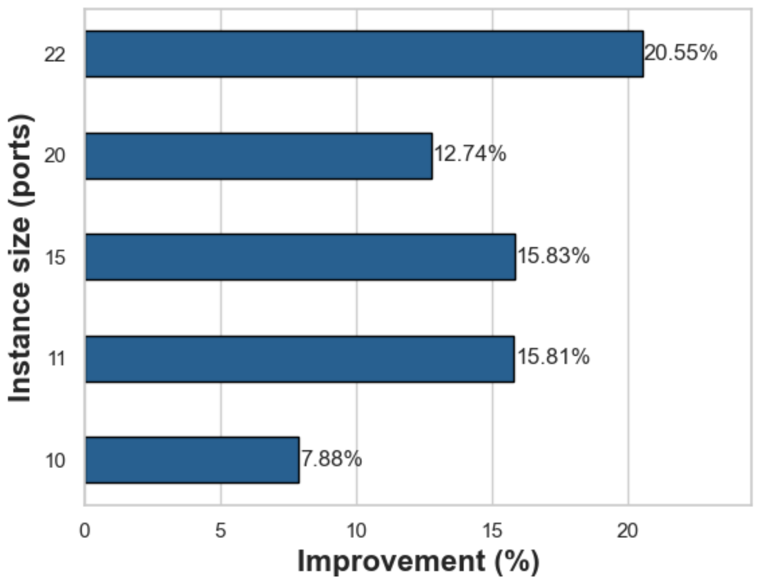


Figure 7.1: DRLH improvement over AH on ABC instances

Figure 7.1 shows the average improvement of DRLH with AH as the baseline. It shows how well DRLH performs based on instance size (number of ports). 7.1 indicates that DRLH favors bigger instances and are able to outperform AH on average.

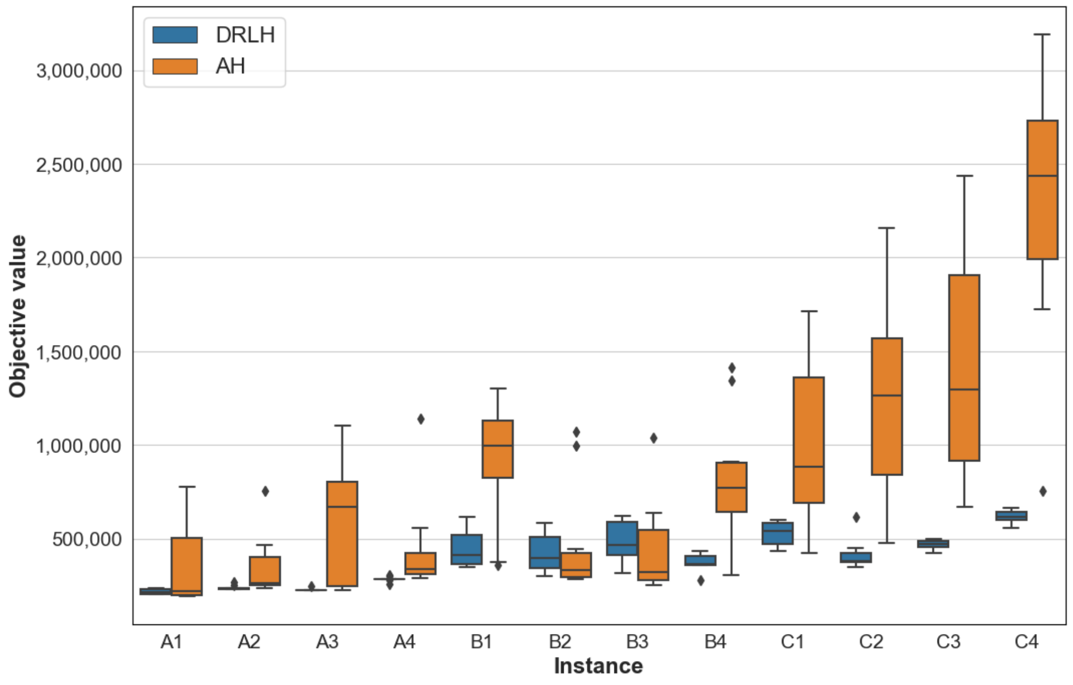


Figure 7.2: Boxplot of AH and DRLH performance on the ABC instances for FNDP-OT

Figure 7.2 shows how well DRLH and Adaptive Heuristic (AH) perform over 10 iterations. We can clearly see that DRLH is overall more consistent. We can see this by the size of the box itself, which represents the Interquartile Range (middle 50% of the results). We can also see that DRLH has fewer outliers than AH.

7.2 Results of FNDP-OT with OD-patterns

The results presented in this section are strictly bound to Feeder Network Design Problem with Origin-Destination. This means that *speed optimization* for the vessels and *OD-patterns* are now included in each instance. DRLH and AH was applied to 16 instances with a number of ports ranging from 10 to 33. The initial costs ranged from 1,787,425.02 to 15,127,695.00. Each algorithm was executed 10 times on each instance to ensure comprehensive and robust results.

7.2.1 ABC instances

Table 7.3 represents the efficacy of the DRLH when applied to the ABC instances with OD-patterns. Upon initial examination, it is apparent that the DRLH significantly reduces costs across all instances. With an average improvement of approximately 90.49%, this demonstrates that DRLH successfully optimizes the routes and outperforms AH on all instances.

Table 7.3: Detailed DRLH results on the ABC instances with OD-patterns

Instance	#Ports	Initial	Avg	Best	Improvement	Time
A1	10	1 787 425.02	165 363.82	162 094.97	90.93%	3.84
A2	10	2 283 753.94	228 739.93	215 610.03	90.56%	3.57
A3	11	2 232 435.71	207 252.85	198 126.80	91.13%	3.89
A4	11	2 552 118.45	244 171.33	238 281.07	90.66%	4.02
B1	15	2 914 909.61	306 566.05	263 597.42	90.96%	6.66
B2	15	2 519 664.82	331 637.61	271 610.36	89.22%	6.58
B3	15	2 234 808.65	325 516.42	246 148.24	88.99%	6.56
B4	15	2 785 142.02	333 149.47	265 378.83	90.47%	7.44
C1	20	3 473 140.68	497 320.05	381 250.75	89.02%	15.40
C2	22	3 982 235.83	527 084.34	338 532.52	91.50%	20.33
C3	22	4 674 235.83	531 312.68	411 116.06	91.20%	26.65
C4	22	5 992 235.83	581 222.33	528 084.66	91.19%	27.87
Average					90.49%	11.49

7.2.2 Adapted LINER-LIB instances

Table 7.4 shows similar results as for the ABC instances. We see that DRLH is able to optimize and significantly reduce costs across all instances with an average improvement of 78.43.

Table 7.4: Detailed DRLH results on the adapted LINER-LIB instances with OD-patterns

Instance	#Ports	Initial	Avg	Best	Improvement	Time
BALTIC	12	5 930 582.18	919 099.62	900 190.53	84.82%	8.88
MEDIT-1	33	7 430 958.32	1 722 655.03	1 365 022.73	81.63%	59.71
MEDIT-2	32	4 110 291.35	1 210 160.27	1 013 134.49	75.35%	46.22
WAF	20	15 127 695.20	4 454 236.93	4 249 151.49	71.91%	9.60
Average					78.43%	31.10

7.2.3 Performance comparison between DRLH and AH

Table 7.5 shows that DRLH outperforms AH on 14 out of 16 instances for the best-found objective. But on average, DRLH outperforms the AH, often by a substantial amount on all instances. The average improvements range from 218,693,23 (A1) to as high as 4,334,118.77 (WAF).

Table 7.5: Comparison of results on the ABC and adapted LINER-LIB instances with OD-patterns

Instance	#Ports	AH	DRLH	Δ Best	Δ Avg
A1	10.0	169 459.4	162 095.0	7 364.4	218 693.2
A2	10.0	207 296.0	215 610.0	-8 314.0	471 330.6
A3	11.0	383 765.5	198 126.8	185 638.7	688 309.3
A4	11.0	279 473.6	238 281.1	41 192.5	617 463.6
B1	15.0	358 309.8	263 597.4	94 712.4	690 225.1
B2	15.0	360 605.0	271 610.4	88 994.7	390 064.6
B3	15.0	232 405.5	246 148.2	-13 742.7	294 824.0
B4	15.0	292 281.2	265 378.8	26 902.3	678 244.9
C1	20.0	400 497.8	381 250.8	19 247.1	824 840.7
C2	22.0	1 214 101.1	338 532.5	875 568.6	1 236 663.0
C3	22.0	855 917.0	411 116.1	444 800.9	1 926 362.2
C4	22.0	2 807 765.5	528 084.7	2 279 680.8	2 813 572.1
BALTIC	12.0	1 070 335.4	900 190.5	170 144.9	931 553.3
MEDIT-1	33.0	3 042 545.2	1 365 022.7	1 677 522.5	2 100 478.9
MEDIT-2	32.0	1 789 563.4	1 013 134.5	776 428.9	835 325.2
WAF	20.0	6 997 154.8	4 249 151.5	2 748 003.3	4 334 118.8
Average		1 278 842.3	690 458.2	588 384.1	1 190 754.3

In Figures 7.4 and 7.5, we can clearly see that DRLH improves drastically compared to AH on all instance sizes. Figure 7.3 shows how consistently DRLH performs compared to AH.

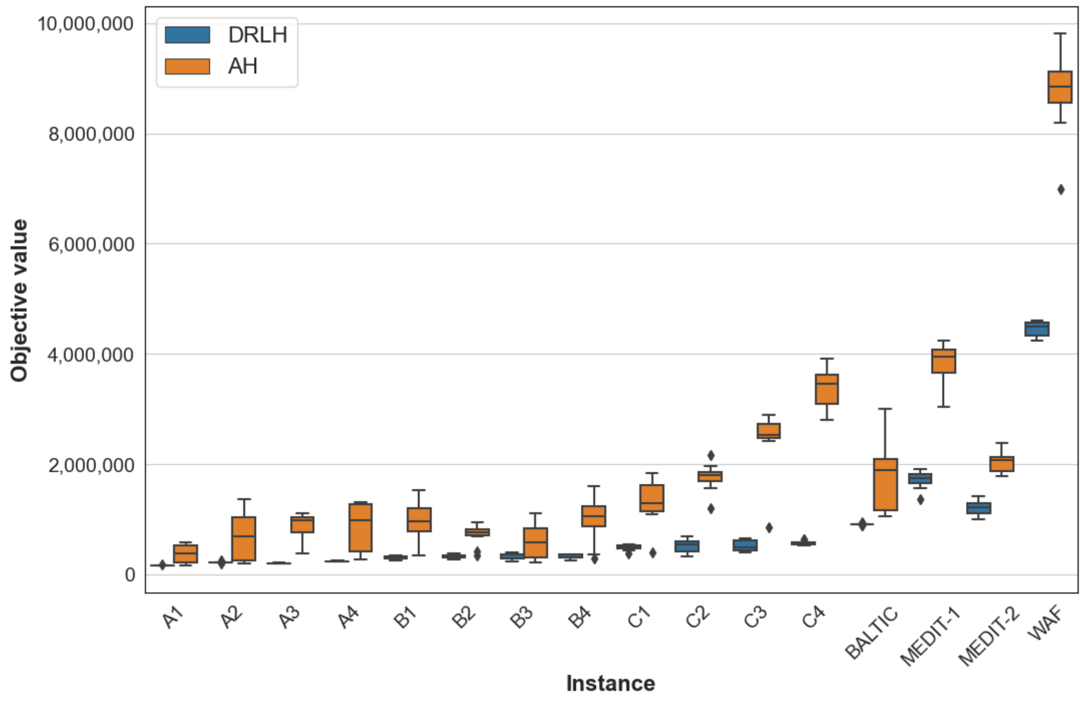


Figure 7.3: Boxplot of AH and DRLH performance on the ABC and adjusted LINER-LIB instances for FNDP-OT with OD-patterns

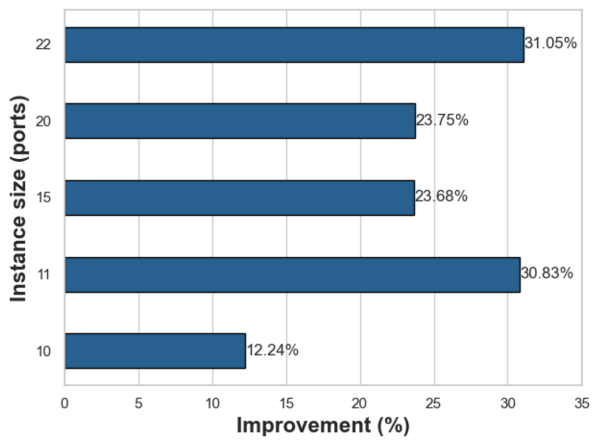


Figure 7.4: DRLH improvement over AH on the ABC instances with OD-patterns

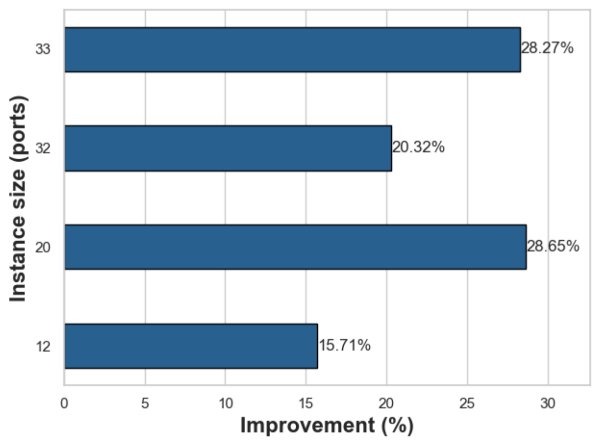


Figure 7.5: DRLH improvement over AH on adapted LINER-LIB

Performance on different sizes

Figure 7.6 displays how AH and DRLH performes based on instance size, and DRLH outperforms AH on all instances. There is also clear that DRLH is a lot more persistent

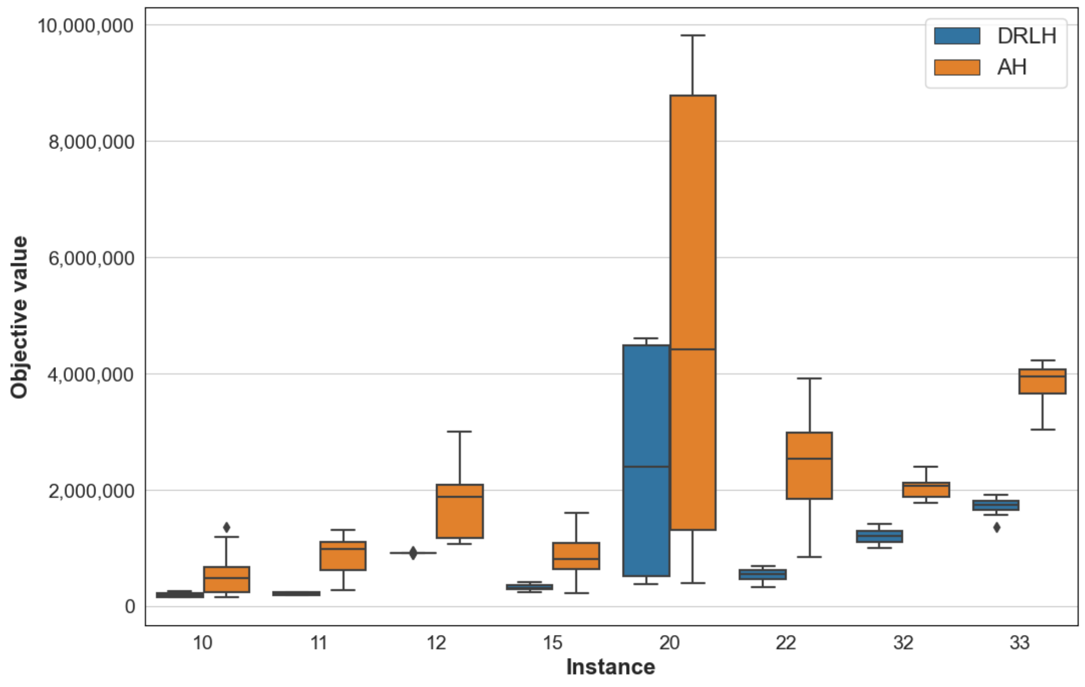


Figure 7.6: Boxplot of AH and DRLH performance on different instance sizes for FNDP-OT with OD-patterns

on average and therefore a more trustworthy approach compared to AH, by consistently delivered higher-quality solutions.

7.3 Heuristic Selection Strategies

The disparity in action selection probabilities between DRLH and AH is notable. In Figure 7.7, the action probabilities of AH are illustrated during the search process, for instance, A1-OD. This differs from the smoothed values for DRLH, as shown in Figure 7.8.

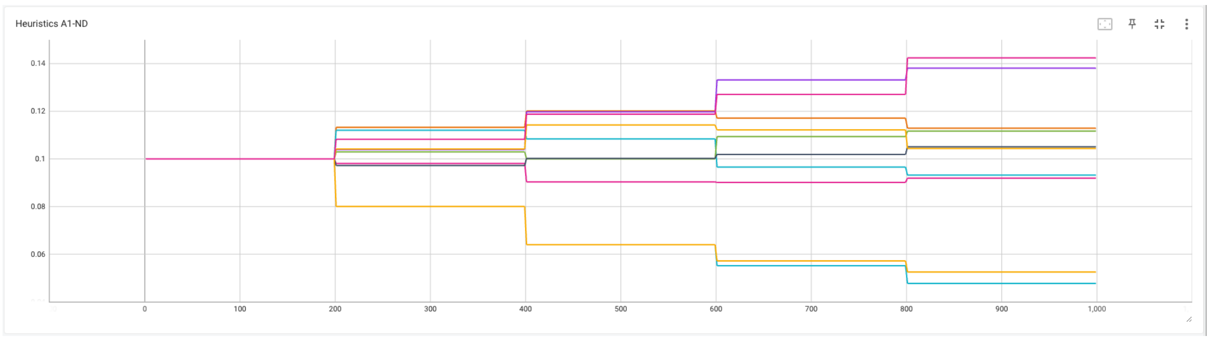


Figure 7.7: Actions selection probabilities for Adaptive Heuristic

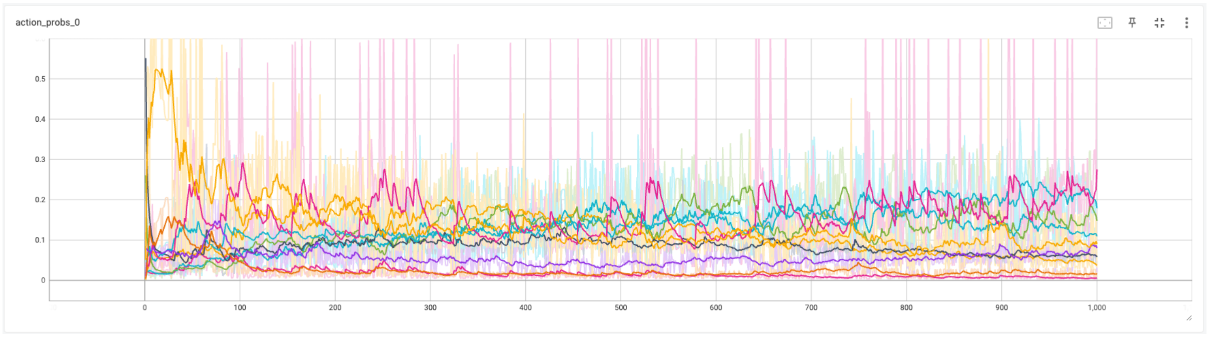


Figure 7.8: Action selection probabilities for DRLH with applied smoothing

We can see the unique advantage of DRLH lies in its ability to micro-manage the heuristics, enabling precise and quick modifications from iteration to iteration. This dynamic is powered by the combined strength of Deep Learning and Reinforcement Learning, which enables DRLH to make the most of the state information available at each search stage. Consequently, DRLH can optimize the usage of each operator more effectively and manage a broader pool of heuristics.

Although both the meta-heuristic and the hyperheuristic display a tendency to prioritize specific heuristics over others, updating weights in each segment by AH fails to provide the adaptability DRLH offers. This results in AH being less responsive to the particularities of the solution space it operates within. This demonstrates that DRLH is better at adapting to the state of the solution, exploring the neighborhood, and thereby maximizing its overall performance. This distinction is evident in Figure 7.9, where the variation in ALNS is less pronounced compared to AH. Furthermore, Figure 7.10 provides additional support, clearly demonstrating that DRLH effectively utilizes the micro-level heuristics from the beginning, allowing it to navigate the neighborhood more effectively and ultimately works towards better solutions.

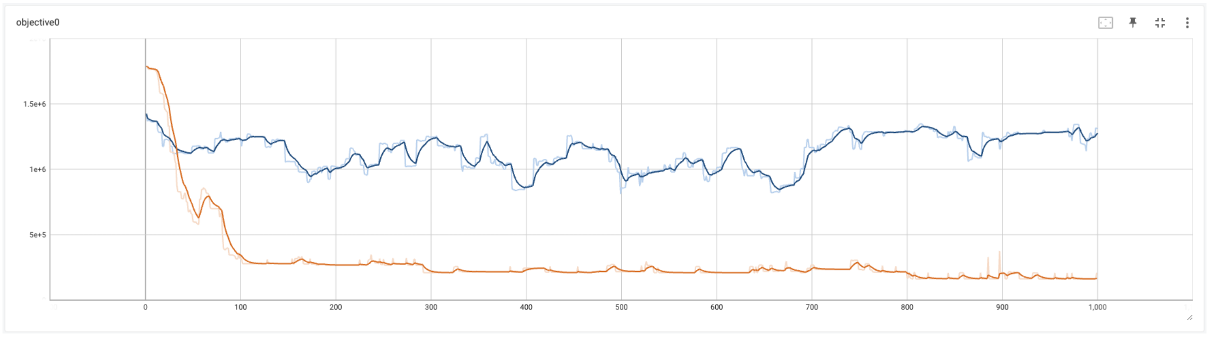


Figure 7.9: The objective cost of the incumbent over the search for instance A1 (DRLH: yellow, AH: blue)

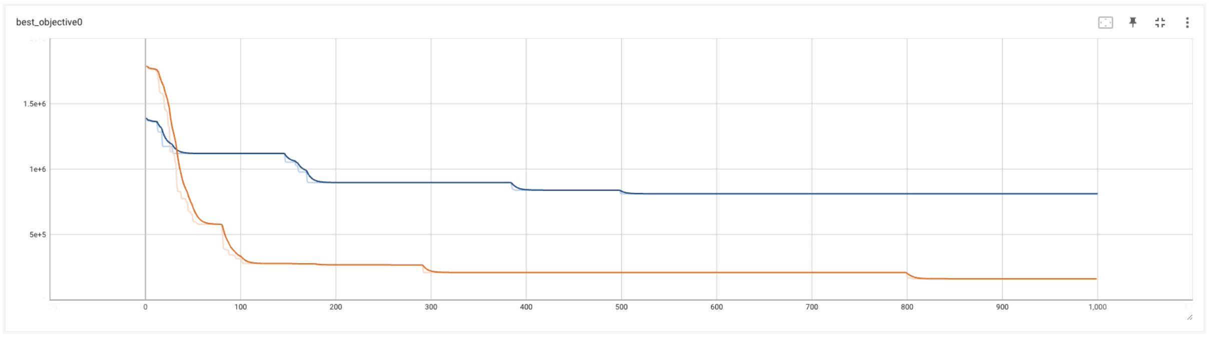


Figure 7.10: The best-found objective over the search for instance A1 (DRLH: yellow, AH: blue)

We can also observe that the AH initiates with both a lower initial objective and a lower best objective compared to the DRLH. This outcome is a result of the tune iterations conducted by AH. While adjusting the temperature for the acceptance criterion, we keep a record of the incumbent and the best solution identified during this process.

Chapter 8

Conclusion and Future Work

In this thesis, we conducted analytic research on two different methods for solving the complex Feeder Network Design Problem with Optional Transshipment and FNDP-OT with OD-patterns. One method is tailored to specific problems and utilizes weight distribution for the selection of low-level heuristics. The other is a general hyperheuristic framework that utilizes the power of Deep Reinforcement Learning for the selection of low-level heuristics to be applied to the solution. In our experiments, we showed that even though the Adaptive Heuristic approach is tailored to the specific problem, the Deep Reinforcement Learning Hyperheuristic approach is able to achieve better results for a substantial amount of the problem instances, with some exceptions. We also demonstrated that DRLH is able to search the neighborhood more efficiently and utilizes the low-level heuristics better compared to AH, resulting in a more stable and reliable approach.

Future research should focus on providing a more extensive set of low-level heuristics specifically designed for the FNDP-OT and FNDP-OT-OD problems. Additionally, implementing simple removal and insertion operators tailored to these problems could further enhance the DRLH agent's ability to exploit the problem structure effectively. It would be interesting to compare the performance of DRLH and AH on larger instances. Specifically, tests could be conducted with extended searches of up to 20,000 iterations, as this is the number of iterations under which the original AH was tested.

It would also be valuable to explore other variants of reinforcement learning algorithms or advanced variations of Deep Reinforcement Learning to identify potential improvements in the solution quality and computational efficiency. By advancing and refining these techniques, one can make significant progress in solving the Feeder Network Design Problem with Optional Transshipment and its OD variant.

List of Acronyms and Abbreviations

AC Actor-Critic.

ACO Ant Colony Optimization.

AH Adaptive Heuristic.

ALNS Adaptive Large Neighborhood Search.

CO Combinatorial Optimization.

COP Combinatorial Optimization Problem.

CVRP Capacitated Vehicle Routing Problem.

DL Deep Learning.

DRL Deep Reinforcement Learning.

DRLH Deep Reinforcement Learning Hyperheuristic.

FEU Forty-foot equivalent unit.

FNDP Feeder Network Design Problem.

FNDP-OT Feeder Network Design Problem with Optional Transshipment.

FNDP-OT with OD-patterns Feeder Network Design Problem with Origin-Destination.

GA Genetic Algorithm.

IQR Interquartile Range.

KP Knapsack Problem.

LNS Large Neighborhood Search.

LSNDP Liner Shipping Network Design Problem.

MST Minimum Spanning Tree.

OD Origin-Destination.

PDP Pickup and Delivery problem.

PDPTW Pickup and Delivery Problem with Time Windows.

PG Policy Gradient.

PJSP Parallel Job Scheduling Problem.

PPO Proximal Policy Optimization.

RL Reinforcement Learning.

SA Simulated Annealing.

SP Scheduling Problem.

TC rate time charter rate.

TEU Twenty-foot equivalent unit.

TS Tabu Search.

VRP Vehicle Routing Problem.

Bibliography

Liner shipping: The backbone of world trade. Online. URL https://www.worldshipping.org/statements/liner-shipping-the-backbone-of-world-trade.

Morten Bergmann, Mohamed Kais Msakni, Ahmad Hemmati, and Kjetil Fagerholt. An adaptive heuristic for feeder network design with optional transshipment. *Transportation Research Part E: Logistics and Transportation Review*, 176:103153, 2023. ISSN 1366-5545. doi: https://doi.org/10.1016/j.tre.2023.103153. URL https://www.sciencedirect.com/science/article/pii/S1366554523001412.

Berit D. Brouer, J. Fernando Alvarez, Christian E. M. Plum, David Pisinger, and Mikkel M. Sigurd. A base integer programming model and benchmark suite for liner-shipping network design. *Transportation Science*, 48(2):281–312, 2014. ISSN 00411655, 15265447. URL http://www.jstor.org/stable/43666638.

E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward. *A Classification of Hyper-heuristic Approaches*, pages 449–468. Springer US, Boston, MA, 2010. ISBN 978-1-4419-1665-5. doi: 10.1007/978-1-4419-1665-5_15. URL https://doi.org/10.1007/978-1-4419-1665-5_15.

Peter Cowling, Graham Kendall, and E Soubeiga. A hyperheuristic approach to scheduling a sales summit. pages 176–190, 01 2001. ISBN 3-540-42421-0.

Y. Crama and M. Schyns. Simulated annealing for complex portfolio selection problems. *European Journal of Operational Research*, 150(3):546–571, 2003. ISSN 0377-2217. doi: https://doi.org/10.1016/S0377-2217(02)00784-1. URL https://www.sciencedirect.com/science/article/pii/S0377221702007841. Financial Modelling.

H. Fisher and G. Thompson. *Probabilistic learning combinations of local job-shop scheduling rules*, pages 225–251. Prentice-Hall, Englewood Cliffs, 1963.

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`.

Qiang Meng, Shuaian Wang, Henrik Andersson, and Kristian Thun. Containership routing and scheduling in liner shipping: Overview and future research directions. *Transportation Science*, 48(2):265–280, 2014. ISSN 00411655, 15265447. URL `http://www.jstor.org/stable/43666637`.

Mohamed Kais Msakni, Kjetil Fagerholt, Frank Meisel, and Elizabeth Lindstad. Analyzing different designs of liner shipping feeder networks: A case study. *Transportation Research Part E: Logistics and Transportation Review*, 2020. doi: https://doi.org/10.1016/j.tre.2020.101839. URL `https://doi.org/10.1016%2Fj.tre.2020.101839`.

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.

A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume B. Springer, 2003.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Lecture notes in computer science*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. ISBN 9783540652243.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking, 2022.

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

Richard S. Sutton. *Reinforcement learning : an introduction.* Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition. edition, 2018. ISBN 0-262-35270-2.

Appendix A

Experiments on Different Reward Functions

A.1 R_t^{5310}

$$R_t^{5310} = \begin{cases} 5 & \text{if } f(s') < f(s_{best}) \\ 3 & \text{if } f(s') < f(s) \\ 1 & \text{if } accept(s', s) \\ 0 & \text{otherwise} \end{cases} \tag{A.1}$$

The R_t^{5310} reward function focuses on exploration while encouraging exploitation with a higher reward. One issue regarding this is reward hacking. For particular instances, the R_t^{5310} is vulnerable to reward hacking, i.e., the agent learns that by continuously improving the solution by small amounts, it receives more cumulative reward over the search than by intensifying heavily. This is why we had to utilize early stoppage on the smaller instances to retrieve the model before this.

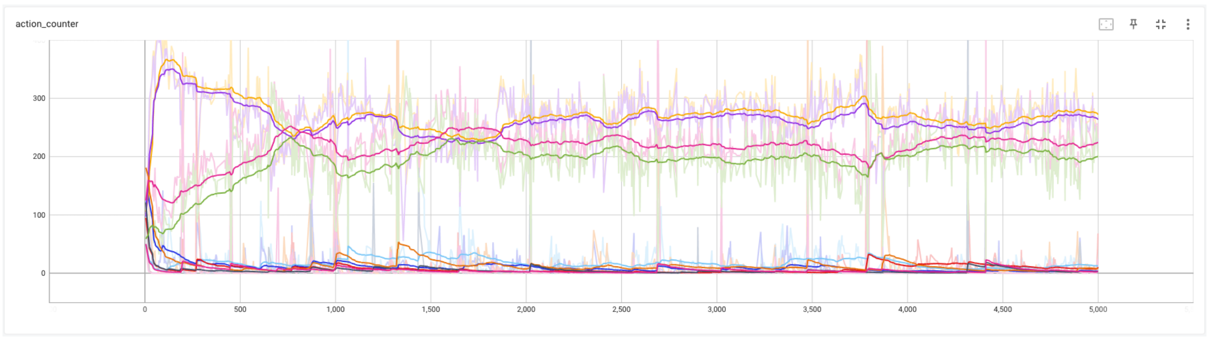


Figure A.1: Reward function R_t^{5310}

This reward function heavily favors some of the operators, but as we can see, it utilizes the entire pool. Figure A.1 illustrates the selection count of each heuristic through the search.

A.2 Improvement-based reward functions

$$R_t^{PM} = \begin{cases} 1 & \text{if } f(s') < f(s_{best}) \\ -1 & \text{otherwise} \end{cases} \tag{A.2}$$

$$R_t^{AX} = \begin{cases} \Delta & \text{if } f(s') < f(s_{best}) \\ 0 & \text{otherwise} \end{cases} \tag{A.3}$$

$$R_t^{BX} = \begin{cases} \dfrac{\Delta}{100000} & \text{if } f(s') < f(s_{best}) \\ 0 & \text{otherwise} \end{cases} \tag{A.4}$$

$$R_t^{CX} = \begin{cases} \Delta & \text{if } f(s') < f(s_{best}) \\ \dfrac{\Delta}{2} & \text{otherwise} \end{cases} \tag{A.5}$$

These functions are introduced since it was a part of the selection process when solving the FNDP-OT with OD-patterns. The reward functions were trained on all instance sizes with 5 000 random instances and tested on the test instances, ABC, and adjusted LINER-LIB instances. By doing so, we were able to find useful patterns in the data.

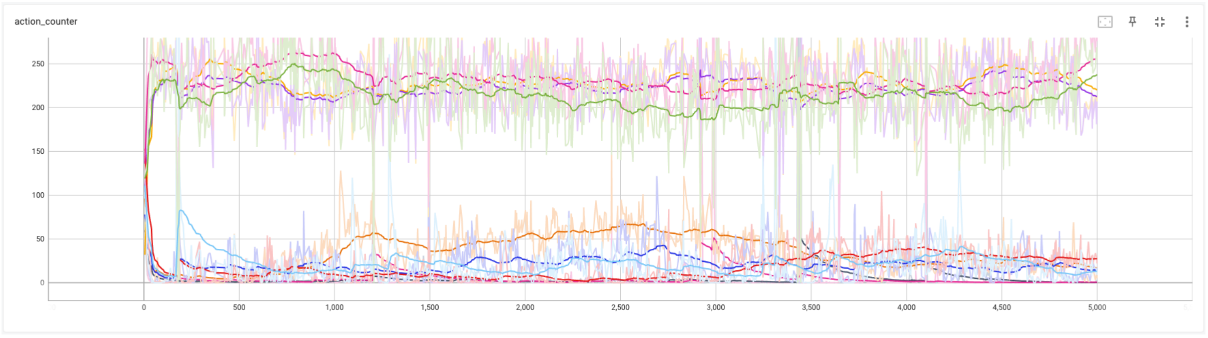


Figure A.2: The R_t^{PM} action probabilities over 1k iterations on 5000 random instances

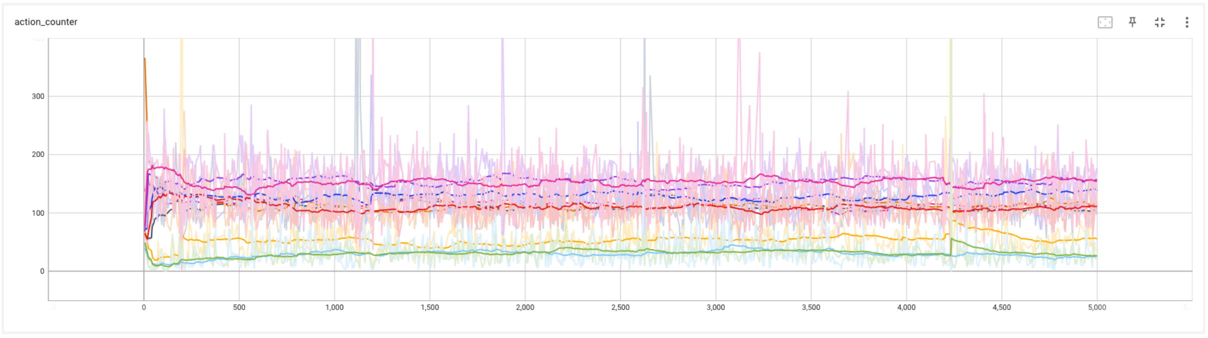


Figure A.3: The R_t^{AX} action probabilities over 1k iterations on 5000 random instances

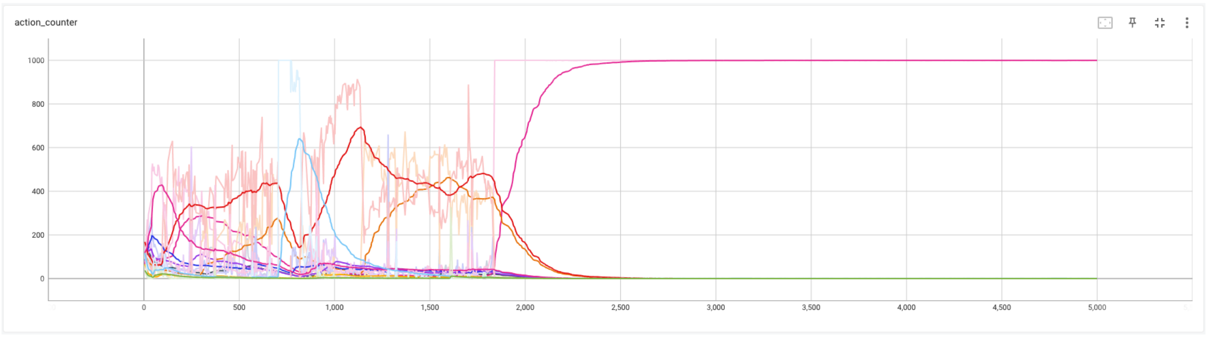


Figure A.4: The R_t^{BX} action probabilities over 1k iterations on 5000 random instances

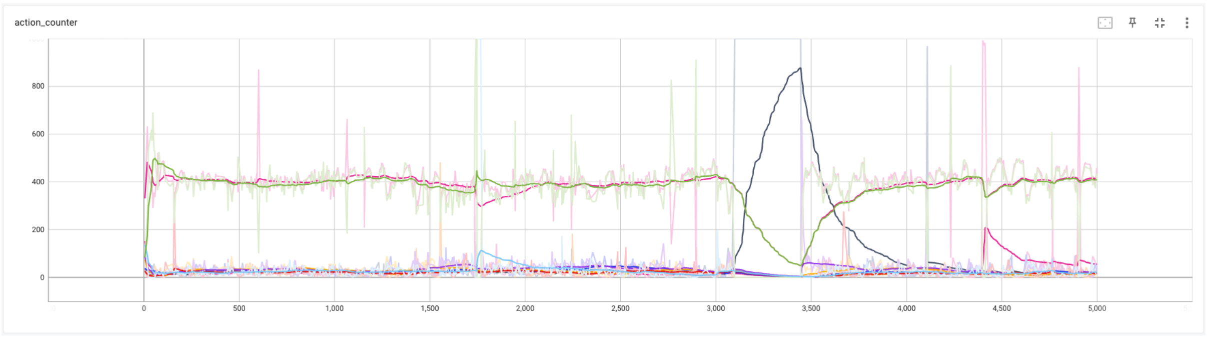


Figure A.5: The R_t^{CX} action probabilities over 1k iterations on 5000 random instances

The line graphs depicted in Figures A.2, A.3, A.4, and A.5 provide a comprehensive view of the heuristic selection probabilities during the agent's training phase on the FNDP-OT with OD-patterns. It is evident from these visual representations that the inclination towards specific heuristics varies significantly depending on the reward functions; some display a pronounced preference for certain heuristics, while others distribute their choices more evenly.

Performance

An intriguing correlation surfaces when we compare Figure A.6, which shows the best solution objective discovered across all 5,000 instances, with Figure A.7, representing the number of distinct solutions encountered during the search. This parallelism underscores the efficiency of the agent's ability to navigate the neighborhood in the solution space, highlighting its ability to balance exploration and exploitation.

The reward functions has the corresponding color:
R_t^{5310}: Black, R_t^{AX}: Blue, R_t^{BX} Green, R_t^{CX}: Pink, R_t^{PM}: Purple.

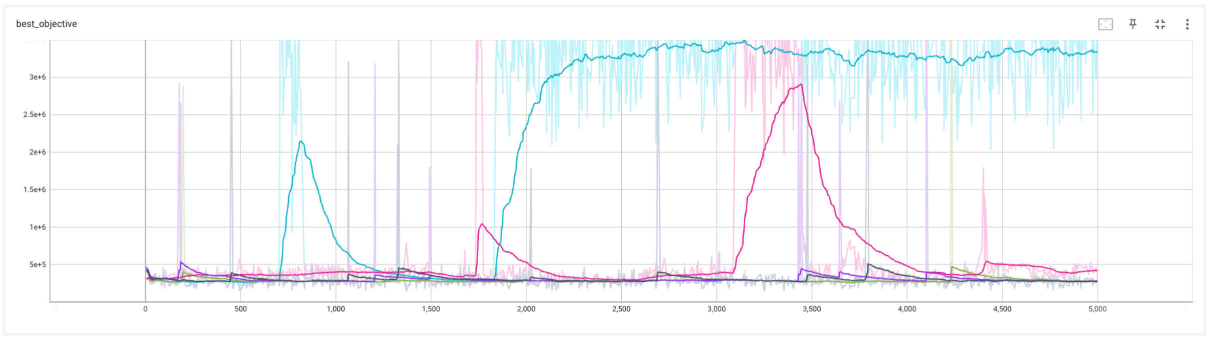


Figure A.6: The best objective found for all reward functions on the 5000 instances

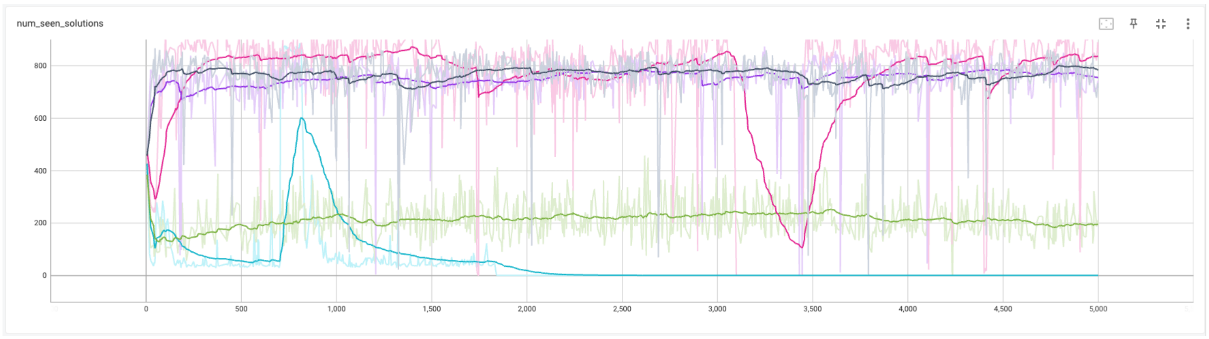


Figure A.7: The number of solutions explored for all reward functions on the 5000 instances

Figures A.6 and A.7 illustrate that R_t^{BX} fails to converge to an effective policy. This outcome is likely attributed to its tendency to significantly diminish the reward received upon making substantial improvements. Consequently, R_t^{BX} is unable to learn since it primarily relies on the *enable_demand* heuristic (pink A.4). This approach nearly guarantees a constant reward, as it is the heuristic most likely to ensure an improved feasible solution compared to the incumbent. As a result, R_t^{BX} demonstrates a minimal exploration of the state space in comparison to the other reward functions.

The differing impact of the reward functions on exploration is evident. These differences further reflect on the best objective found during the search. Agents trained with the reward functions R_t^{5310}, R_t^{PM}, and R_t^{CX} show improved exploration. Nonetheless, R_t^{CX} reveals a more irregular pattern, suggesting that the agent's policy is adjusted based on environmental feedback. This indicates a possible convergence towards the local minimums.