Deep Reinforcement Learning for Multi-Objective Airport Ground Handling

Sayed Mozafar Shah

Friday August 29, 2025

Deep Reinforcement Learning for Multi-Objective Airport Ground Handling

by

Sayed Mozafar Shah

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday August 29, 2025 at 10:30 AM.

Student number:	4717368	
Thesis committee:	Dr. N. Yorke-Smith,	TU Delft, supervisor
	Prof. W. Yaoxin,	TU Eindhoven, external supervisor
	Dr. J. W. Böhmer	TU Delft, independent examiner

TU Delft

Preface

بسم اللّه الرّحمن الرّحيم

In the name of God, the most gracious, the most merciful.

I want to thank my supervisors Dr. Neil Yorke-Smith and Prof. Yaoxin Wu for their assistance and guidance throughout this project. Additionally, I want to thank the friends I met during my years in Delft, both during my Bachelor's and Master's studies, for making this time of my life memorable and enjoyable. Most importantly, I would like to express my sincerest gratitude to my parents for all the sacrifices they have made in their lives, which allowed me to pursue any degree.

Sayed Mozafar Shah
Delft, August 2025

Abstract

Air transport has enormous impact on economic, social, and environmental factors worldwide. According to the International Air Transport Association significant year on year increases can be noticed recently, in both passenger and cargo traffic. However, with this increasing demand for air transport, airports are faced with a rising trend in congestion, delays, and other problematic inefficiencies. Although increasing demand is a factor in these challenges, airline or airport causes, such as ground handling, remain the second largest cause of delays. This highlights the need for efficient and optimal scheduling of ground handling processes to minimise delays, and thereby the negative impact this might have on the airlines or airports. Most existing studies address only simplified sub-problems of Airport Ground Handling (AGH), by relaxing constraints or by leaving them out, rather than tackling the complete problem. Furthermore, these approaches tend to focus on single objective optimisation of AGH, while in practice there can be many (conflicting) objectives that need to be optimised simultaneously. This research explores the possibility of extending a neural model, which is trained to optimise instances of AGH on a single objective, with a generic learning-based approach that approximates the Pareto set for multi-objective optimisation, and applying further hypothetical improvements to this combined model. The implemented models are compared against each other, heuristic approaches for multi-objective combinatorial optimisation, and against the original single-objective neural model.

Contents

List of Figures

List of Tables

1 Introduction

Over the entire world air traffic has a tremendous economic, social, and environmental impact. According to the International Air Transport Association, there has been a year on year increase in both global passenger and cargo traffic, measured in October 2024. The global Cargo Tonne-Kilometers grew by 9.8%, which was the 15th consecutive month of growth [1]. While total domestic passenger traffic grew by 3.5%, and international passenger traffic increased by 9.5% [2]. With the growth of air traffic in recent years, a rising trend in the number of congested airports and flight delays were noticed [3]. There was an average delay of 17.3 minutes in 2022, which was at a 5-year high, almost doubling from the year before. Further, the number of flights increased by 48% compared to 2021 [4]. Although these delays were also the consequence of increasing number of passengers and flights following the declines of the COVID-19 pandemic, airline and airport related causes increased as well. Airline causes, such as ground handling, ranked second on causes that contributed most to this average delay, with 4.9 minutes per flight [4]. Hence, ground handling operations have a significant impact on delays. Therefore, efficient and accurate scheduling tools could aid in reducing these delays, and the related economic loss, at an airport. In order to come up with helpful solutions there must a well defined problem, which can then be attempted to be solved.

To define the problem, one must know the processes involved in *airport ground handling* (AGH). Ground handling involves various processes or operations, which need to be scheduled in a manner that takes their precedence relations, time windows, and other constraints into account. These operations are needed to prepare the airplane for its next flight. They are always conducted by a fleet of vehicles, which is why AGH is often modelled as a *vehicle routing problem* (VRP) (or a variant thereof) with additional constraints such as capacity, time windows, and precedence relations [5]. In addition to these constraints, the scale of the problem, especially for airports with lots of traffic, is what makes AGH much more complex than conventional VRPs. The Vehicle Routing Problem is a famous NP-Hard combinatorial optimisation problem (COP), with these added constraints, AGH is an extension to VRP, and even to VRP with Time Windows (VRPTW) as AGH involves precedence relations and capacity constraints in addition to the time windows. Although VRPs are often formulated as single-objective optimisation problems, real-world instances of VRP(TW), and many other (combinatorial) optimization problems, involve multiple objectives that need to be optimised simultaneously. This classifies them as multi-objective (combinatorial) optimisation problems (MO(CO)Ps). Therefore, AGH is also a multi-objective combinatorial optimisation problem (MOCOP).

In a general sense, multi-objective (combinatorial) optimisation problems can be formulated as follows:

$$\min_{x \in X} F(x) = [F_1(x), F_2(x), ..., F_n(x)] \tag{1.1}$$

where $F(x)$ consists of n different objective functions, and $X \subseteq \mathbb{R}^d$ represents the search space. Usually, there does not exist a single solution that optimises all objectives simultaneously, because these objectives can conflict with each other. Therefore, various solutions with a different trade-off among the objectives are considered. This set of solutions is referred to as the Pareto set, which is essentially a set of so-called Pareto optimal solutions (a detailed explanation about Pareto optimal solutions is given in Section 2.5.1). Finding even a single one of these solutions can be NP-hard for many problems, and the number of Pareto optimal solutions could be exponentially large with respect to the size of the problem [6]–[9]. Thus, finding all the exact Pareto optimal solutions for a MOCO problem is extremely difficult and time consuming.

Therefore, the goal of this work is to train a neural model capable of solving bi-objective AGH, a real-world MOCO problem, which has not been researched extensively. The model should be able to approximate the Pareto front for any instance of AGH by considering various trade offs amongst the objectives. Rather than having to manually construct a model and its corresponding operators for each type of algorithm, as is the case with meta-heuristic approaches, the goal is to have a neural model that is able to generalise well such that no manual operations are needed. Additionally, the meta-heuristic approaches often require extensive domain knowledge to come up with operators or models that are optimised for the problem to be solved. With a neural model that is able to generalise there is also no (extensive) domain knowledge required. To achieve this, existing approaches for other problems or other variants of AGH are considered, combined, and extended.

1.1. Problem Description

Airport Ground Handling involves various processes or operations, which need to be scheduled in a manner that takes their precedence relations, time windows, and other constraints into account. These operations are needed to prepare the plane for its next flight. There exist various models of AGH, some that relax constraints or leave out constraints altogether. Some, for example, only consider the precedence relations, but no time window and capacity constraints. Whereas others consider the time window constraints, but ignore the other constrains. In this work, the complete AGH problem is considered, with all 3 constraints, i.e. the precedence relations of the operations, the time windows, and the capacity constraints of the vehicles. Since the ground handling operations are always conducted using a fleet of vehicles, the problem is modelled as a Capacitated Vehicle Routing Problem with Time Windows and precedence relations (CVRPTW with precedence relations). Wu, Zhou, Xia, *et al.* [5] also consider the complete AGH problem, but focus solely on single-objective optimisation. However, the MILP model formulated in the paper, in which the AGH problem is described as a multiple-fleet VRP with the additional constraints, will be used as a baseline in this research. This model is formulated by equations 1.2-1.13.

The authors represent the AGH problem as an undirected graph $G = (N, E)$, with node set $N = \{0, 1, ..., n, \dot{n}\}$ and edge set $E = \{(i, j) | i, j \in N; i \neq j\}$. The nodes are used to represent locations and the edges are routes connecting two nodes. Both node 0 and node $\dot{n}$ denote the depot node, but the authors use both to differentiate between the starting location of the vehicles and the destination to avoid temporal conflicts at the depot node. $N^* = N \setminus \{0, \dot{n}\}$ denotes the set of flights that need to be served, with a demand δ_i^f for each flight $i \in N^*$ and operation $f \in F = \{1, ..., F\}$. The demand of the depot nodes is set to 0 as these do not have to be served. For any operation $f \in F$, each edge (i, j) is assigned a cost c_{ij}^f. The cost of going from node 0 to node $\dot{n}$ is set to $+\infty$ to prevent meaningless travel between node 0 and $\dot{n}$. To represent the precedence relation between two different operations the operator $\prec$ is used, so $f_1 \prec f_2$ for any $(f_1, f_2 \in F)$ means that operation f_1 precedes f_2. Each fleet corresponding to some operation f exists of a set of vehicles (i.e. a fleet) $V^f = \{1, ..., V^f\}$, with the capacity denoted by Q^f. Additionally, d_i^f is used to denote the duration of performing operation f for flight i. For any operation f the duration of nodes 0 and $\dot{n}$ is set to 0. Furthermore, they use t_ij^f to denote the travel time from flight i to j by some vehicle for operation f, with $t_{0\dot{n}}^f = 0, \forall f \in F$. The variables a_i^f and b_i^f denote the time window for the start time of operation f for flight i, i.e. the start time for operation f on flight i should be within $[a_i^f, b_i^f]$ [5]. This formulation is extended to include the second objective to be optimised in combination with the necessary constraints to ensure the correctness of the values of this objective, defined in equations 1.14-1.15.

$$\min. \sum_{f \in F} \sum_{v \in V^f} \sum_{(i,j) \in E} c_{ij}^f x_{ijv}^f \tag{1.2}$$

$$\text{s.t.} \sum_{i \in N} \sum_{v \in V^f} x_{ijv}^f = 1, \forall j \in N^*, f \in F \tag{1.3}$$

$$\sum_{i \in N \setminus \{\dot{n}\}} x_{iuv} = \sum_{j \in N \setminus \{0\}} x_{ujv}^f, \forall u \in N^*, v \in V^f, f \in F \tag{1.4}$$

$$\sum_{j \in N^*} \sum_{v \in V^f} x_{0jv}^f \leq V^f, \forall f \in F \tag{1.5}$$

$$\sum_{j \in N^*} \sum_{v \in V^f} x_{0jv}^f = \sum_{i \in N^*} \sum_{v \in V^f} x_{i\dot{n}v}^f, \forall f \in F \tag{1.6}$$

$$\sum_{i \in N \setminus \{0\}} \sum_{v \in V^f} x_{i0v}^f = \sum_{j \in N \setminus \{\dot{n}\}} \sum_{v \in V^f} x_{\dot{n}jv}^f = 0, \forall f \in F \tag{1.7}$$

$$\sum_{i \in N^*} \delta_i^f \sum_{j \in N} x_{ijv}^f \leq Q^f, \forall v \in V^f, f \in F \tag{1.8}$$

$$x_{ijv}^f (T_{iv}^f + d_i^f + t_{ij}^f - T_{jv}^f) \leq 0, \forall v \in V^f, f \in F \tag{1.9}$$

$$a_i^f \leq T_{iv}^f \leq b_i^f, \forall v \in V^f, f \in F \tag{1.10}$$

$$T_{iv}^{f_1} + d_i^{f_1} \leq T_{iv}^{f_2}, \forall f_1, f_2 \in F, f_1 \prec f_2 \tag{1.11}$$

$$x_{ijv}^f \in \{0, 1\}, \forall (i,j) \in E, v \in V^f, f \in F \tag{1.12}$$

$$T_{iv}^f \geq 0, \forall i \in N^*, v \in V^f, f \in F \tag{1.13}$$

The model is extended by adding another objective function that should be minimised, namely the latest (total) completion time. This extension is necessary to formulate a second objective for the bi-objective optimisation of the AGH problem. For this extension we define another variable named C_{max}, which represents the completion time of the latest operation in the entire schedule, i.e. the global makespan of all operations across all flights. For the corresponding objective, this variable needs to be minimised, so another objective function will be added to the problem model defined as $\min C_{max}$ in equation 1.14. However, this variable must be constrained correctly to ensure that its value correctly represents the latest completion time. To that end, the variable is constrained as defined in equation 1.15, which effectively states that for every flight i and every operation f on that flight C_{max} has to be no less than the maximum over the time at which we can start serving (T_{iv}^f) flight i by vehicle v for operation f added by the service time (d_i^f) for flight i with regards to operation f. This directly forces C_{max} to be at least as large as every operation's completion time over all the flights, ensuring that it represents the latest completion time over the entire schedule.

$$\min. C_{max} \tag{1.14}$$

$$\text{s.t. } C_{max} \geq T_{iv}^f + d_i^f, \forall i \in N^*, \forall v \in V^f, \forall f \in F \tag{1.15}$$

1.2. Contribution

This thesis will specifically focus on solving a bi-objective variant of the global AGH problem. The two objectives that will be optimised will be the ***global travel distance of all vehicles in all operations***, and the ***latest (total) completion time***. For both objectives, it holds that a smaller value is preferred, i.e. the objectives have to be minimised. The multi-objective (MO) AGH problem (or its equivalent VRP(TW)) is not researched as extensively as other instances of VRP, such as TSP. This thesis contributes to the research by providing a novel perspective on solving MO AGH instances using a neural approach, and possibly outperforming existing solutions. For this contribution it tries to answer the research questions mentioned in section 1.3.

1.3. Research Questions

This report focusses on the following research questions:

1. Are the objectives, i.e. total travel distance and maximum completion time, interdependent?
   (a) Does optimising for one of the objectives lead to an optimal solution for the other objective?
   (b) Can these objectives be optimised independently, i.e. can we solve for the respective single-objective variants and combine the optimal solutions for the objectives without needing to make a trade-off?
2. What solutions, using the Pareto front approach, exist for solving the Multi-Objective AGH problem?
   (a) Which ones can be used for learning-based approaches?
   (b) Do these solutions use exact methods or approximation methods?
   (c) If approximation methods are used, what metrics will be used to determine the accuracy?

1.4. Overview

This thesis report is structured as follows. First, the necessary background information needed to understand the terminology in this thesis report will be provided in Chapter 2, which also includes the related works. Afterwards, the methodology will be explained in Chapter 3, which provides insights on the approach itself and its motivations. In Chapter 4, the conducted experiments and their corresponding results will be discussed. Finally, a conclusion and remarks for future work will be made in Chapter 5.

2 Background & Related Work

This chapter explains the topics relevant to understanding the subjects discussed in this thesis. The chapter starts with introducing what Reinforcement Learning (RL) is, how it works, and what type of problems it is typically applied to. This is followed by section 2.2, in which Deep Learning (DL) is explained, a type of machine learning that utilises multilayered neural networks to extract features or learn complex patterns from input data. Afterwards, the combination of the two techniques, which is known as Deep Reinforcement Learning (DRL), is discussed in section 2.3. This technique is applied in this thesis as well to learn to optimise multi-objective Airport Ground Handling (AGH) instances. After discussing these general learning techniques, theoretical concepts related to Combinatorial Optimisation Problems (COPs) are introduced in section 2.4. The theory related to COPs is extended to Multi-Objective Combinatorial Optimisation Problems, MOCOPs or MOCO Problems for short, in section 2.5. This theoretical knowledge is relevant, because AGH, or its equivalent Vehicle Routing Problem (VRP) with additional constraints, is also a (MO)COP. Therefore, existing approaches to solving such problems are considered as part of the related work discussed in section 2.6.

2.1. Reinforcement Learning

Reinforcement Learning (RL) is one of the three major Machine Learning (ML) paradigms, together with Supervised Learning, and Unsupervised Learning. A tremendous amount of approaches have been implemented and studied within these paradigms [10]–[13]. While needing labelled input-output pairs in the data for a model to learn on with Supervised Learning, Reinforcement Learning does not require this type of presentation of the data [14]. Instead RL enables an agent to learn to approximate an optimal policy for completing a task through directly interacting with its environment [15]. This interaction is done by performing some action A_t while being in some state S_t at a time step t, as depicted in figure 2.1. At each step the agent receives a scalar reward from the environment R_t, which indicates how good the action performed was. The agent is not told which actions to take, but its goal is to maximise the expected cumulative reward received by trying out the defined actions. However, the agent has to consider a trade-off between immediate and delayed rewards [16]. Thus, the agent has to explore the environment through trying actions, but it also has to learn from past exploration attempts, i.e. it has to be able to exploit existing known information to ensure that it is not losing too much reward. In RL this is known as the exploration versus exploitation dilemma. That is where it differs from Unsupervised Learning, in which the goal is typically to find patterns in the provided unlabelled data.

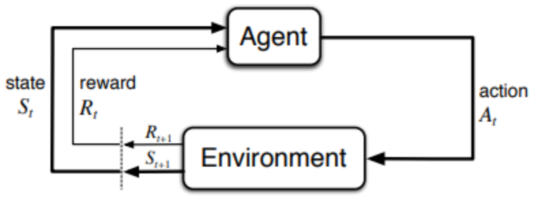


Figure 2.1: The agent-environment interaction in Reinforcement Learning [16]

2.1.1. Markov Decision Process

The high-level diagram, shown in figure 2.1, describes the agent-environment interaction in RL, or more specifically, in a Markov Decision Process (MDP). MDPs are a formal way of modelling sequential decision making problems, in which actions impact both immediate rewards received and future states; thereby influencing future rewards as well. An MDP is defined as a tuple $\langle S, A, P, R, \gamma \rangle$, where:

- S is a finite set of states used to represent the states that exist in some environment
- A is a finite set of actions that the agent can take
- P is a state transition probability distribution, $P^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$, which defines the probability that the agent ends up in some state s' in the next time step given that the agent takes some action a and is in some state s at this time step.
- R is a reward function, which defines the expected reward the agent can get by being in some state s and taking some action a, formally: $R^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- γ is a discount factor between 0 and 1 (including), i.e. $\gamma \in [0, 1]$, which is used to indicate the preference between immediate rewards and future rewards. A gamma value closer to 0 results in short-sighted agent behaviour, whereas a value closer to 1 results in far-sighted behaviour with respect to the received rewards.

MDPs represent an environment in which each state has the Markov Property, which is formally defined as (Equation 2.1):

$$\mathbb{P}[S_{t+1} | S_t] = \mathbb{P}[S_{t+1} | S_1, ..., S_t] \tag{2.1}$$

This essentially means that the current states provides sufficient information about the previous states, such that there is no need to store or use all of the past states. MDPs are incredibly important within the domain of Reinforcement Learning as it can be used to formalise almost all RL problems [17].

2.1.2. Observability

In RL typically a distinction is made in the types of environments based on their observability. An environment is either fully observable or partially observable. The difference between them is determined by the manner in which the agent observes the environment. In a fully observable environment the agent directly observes the environment state, i.e. the agent's state is equal to the observed environment state. Whereas, in a partially observable environment an agent indirectly observes the environment, e.g. a robot having to determine its location based on camera input rather than knowing its location. In these type of environments the agent has to use the available partial observations to infer information about its current state, therefore in such environments it is not the case that the agent's state is equal to the observed environment state. Such environment are formally known as Partially Observable Markov Decision Processes (POMDPs), which theoretically can be converted into MDPs. In partially observable environments the agent has to create its own representation of the environment based on its observations.

Besides the type of environment and the components of an MDP, mentioned in section 2.1.1, RL systems may include a policy, value function, and potentially a model of the environment. The following subsections (subsections 2.1.4 & 2.1.3) explain each of these types in further detail.

2.1.3. Model Based and Model Free RL

Models are used to predict the behaviour of an environment, enabling predictions such as the next possible state or reward the environment might return. They are often used for planning, meaning the agent formulates a sequence of actions in advance by performing computations on its model without any interaction with the actual environment. RL approaches that rely on such models are known as Model Based, while those that learn purely through trial and error, without using a model, are referred to as Model Free. Both Model Based and Model Free approaches use a value function, a policy, or a combination of both.

2.1.4. Value Based and Policy Based RL

A policy, π, defines the agent's behaviour. It is essentially a function that maps a state to an action. It can either be a deterministic policy, i.e. $\pi(s) \rightarrow a$, or a stochastic policy which defines the probability of an action given some state, i.e. $\pi(a|s) \rightarrow \mathbb{P}[A_t = a|S_t = s]$. A value function, v_π, on the other hand, is used as a measure to represent how good or bad it is for an agent to be in a certain state, and therefore it is also used to select between states. Formally, it is defined as the expected future reward the agent may receive if it would be in some state s at a given time t: $v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s]$. Many RL approaches use either a value function or a policy function to determine the actions, and thereby states, of the agents. An RL approach that uses a value function is referred to as Value Based, while those that apply only a policy function are known as Policy Based. There are also RL approaches that utilise both a value function and a policy function, such approaches are named Actor Critic.

There are two types of value functions that are often considered, a state-value function $v_\pi(s)$, and the action-value function $q_\pi(s,a)$. The state-value function defines the expected reward of an MDP starting from state s, and following some policy π, whereas the action-value function defines the expected reward of an MDP starting from state s, and taking action a, then following the policy π. These are known as the Bellman Expectation Equations for v_π and q_π, and they are formulated as shown in equations 2.2 and 2.3, respectively.

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right) \tag{2.2}$$

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s') q_\pi(s',a') \tag{2.3}$$

Notice how both equations contain a component that defines the immediate reward R_s^a the agent gets when taking some action a when in state s, and the discounted value of the next state for the state-value function ($\gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')$), and the discounted value for the next state-action pair in the action-value function ($\gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s') q_\pi(s',a')$).

The goal of the agent is to maximise the cumulative reward, so it has to find some optimal state-value function, or action-value function, over all possible policies π. The optimal state-value function, and action-value function, are defined as $v_*(s) = \max_\pi v_\pi(s)$, and $q_*(s,a) = \max_\pi q_\pi(s,a)$, respectively. However, to understand the behaviour that leads to this maximum cumulative reward, one needs to find the policy that corresponds to the optimal state-value or action-value function. For any MDP, there exists an optimal policy π_* that is better than or equal to all other policies π, i.e. the value function using this policy leads to the optimal value function. Thus, mathematically:

$$\pi_* \geq \pi, \forall \pi \tag{2.4}$$

$$v_{\pi_*}(s) = v_*(s) \tag{2.5}$$

$$q_{\pi_*}(s,a) = q_*(s,a) \tag{2.6}$$

Since a policy is a mapping from state to an action, or some probability distribution that defines the probability of some action given the state, finding an optimal policy π_* is done by performing a maximisation over $q_*(s,a)$ because this contains both action and state. The maximisation is done by defining the probability of $\pi_*(a|s)$ to be 1 whenever an action that maximises the action-value function $q_*(s,a)$ is found, and 0 for the other cases (see equation 2.7 for mathematical representation) [17].

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in A}{\text{argmax}}\ q_*(s,a) \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

The equations 2.2-2.7 provide the foundations to define the Bellman Optimality Equations for v_* and q_*. These optimality equations define the recursive relation of the optimal value functions. These are defined in equations 2.8 and 2.9, respectively.

$$v_*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s') \tag{2.8}$$

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a + \max_{a'} q_*(s',a') \tag{2.9}$$

There exists many iterative solutions to solve these Bellman Optimality Equations, among which Value Iteration, Policy Iteration, Q-learning, and Sarsa.

2.1.5. On and Off-Policy RL

According to Sutton and Barto [16], on-policy RL approaches try to optimise some policy by evaluating and improving the policy that is used to make decisions. In contrast, off-policy RL approaches optimise a policy by evaluating and improving based on some other policy. Q-learning is an example of off-policy learning, in which the agent uses another policy, rather than its own, to select actions greedily during learning. Whereas, SARSA, is an example of an on-policy learning method. In SARSA the same policy, which was used to decide to take some action a at time step t, will also be used to select the next action a' at time step $t+1$.

2.2. Deep Learning

Within the domain of Machine Learning, Deep Learning refers to a neural approach that leverages multi-layered networks. It represents a branch of Machine Learning in which the input data is passed through a hierarchy of layers, each transforming the input into increasingly abstract and complex representations. Generally, it is applied to handle tasks such as classification, regression, and representation learning [18], [19]. The term *deep* in Deep Learning refers to the number of transformation layers that the data passes through. More specifically, Deep Learning models are characterised by large credit assignment path (CAP) depths. The CAP describes the sequence of transformations connecting inputs to outputs, potentially indicating causal relationships. While there is no strict cut-off between shallow and deep learning, most researchers agree that deep learning requires a CAP depth greater than two [20]. A key advantage of Deep Learning is its ability to learn which features to extract and at what level of abstraction. Before, traditional Machine Learning approaches typically relied on manually engineered features to make the data more suitable for classification algorithms and such. Whereas, Deep Learning models can automatically learn feature representations directly from raw data. A visual representation of abstract feature extractions by a Deep Learning Network is represented in figure 2.2.

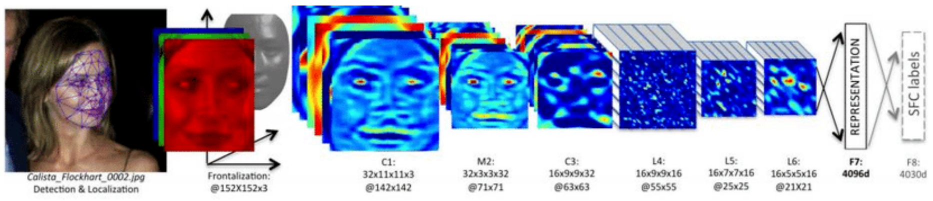


Figure 2.2: Feature extraction pipeline in a Deep Learning Neural Network [21]

However, some manual engineering is still required in Deep Learning, for example, the number of layers and the number of nodes within each layer, and so on [19].

2.3. Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) approaches combine deep learning and RL. DRL enables the integration of the sequential decision making ability of RL with the deep learning networks' ability to learn feature representation as a means to achieve end-to-end learning control capabilities. It is especially useful for RL problems with extremely large state spaces or continuous action spaces because these RL problems used to be intractable using existing approaches. Training a deep neural network to learn the state representation of an environment, then an using an RL agent decisions are made, based on the state representation, on what actions to choose to maximise the cumulative reward. A schematic

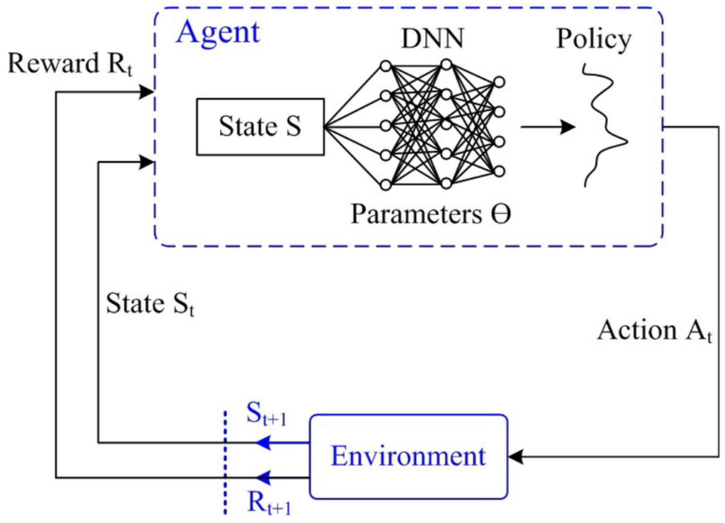


Figure 2.3: Schematic structure of a DRL agent [22]

structure of a general deep reinforcement learning agent is provided in figure 2.3. The deep neural network is trained to learn the state representation, the agent takes actions based on the policy that is optimised based on the state representation learned in the deep neural network, then it receives reward and state feedback from the environment which can be used to further train the deep neural network and optimise the policy using the equations and techniques discussed in section 2.1.4.

2.4. Combinatorial Optimisation Problem

Combinatorial optimisation is a branch of optimisation where the goal is to find the best solution from a finite set of discrete solutions, the size of which is often extremely large. Often, the “best“ solution is defined by some objective function that should be optimised for subject to some constraints, and the solution that results in the highest (or lowest depending on the problem) value is considered to be the best solution. Thus, the solution must be feasible, i.e. adhere to the constraints, and optimise for the objective function. Classic examples of combinatorial optimisation problems (COPs) include the Travelling Salesman Problem (TSP), scheduling, and routing problems, where the goal is to find some optimal combination or ordering of elements [23]. What makes COPs so complex is the combinatorial explosion phenomenon, as the problem size grows, the number of possible solutions increases exponentially, making the solution space too large to search exhaustively using brute force approaches [24]. In fact, many combinatorial optimisation problems are classified as NP-hard, which means that there are no known algorithms that can solve them optimally in polynomial time. This indicates that solving large instances exactly may require a tremendous amount of computational costs. Therefore, other approaches have been introduced to try handle such problems within reasonable time, among which branch-and-bound, or approximation and heuristic methods such as greedy algorithms or meta-heuristics, etc.

As a concrete example of a COP, let us consider the Vehicle Routing Problem (VRP) originally introduced as the Truck Dispatching Problem by Dantzig and Ramser [26]. The objective in VRP is to find an optimal set of feasible routes, for a fleet of vehicles to deliver goods to a given set of customers, typically starting and ending at a depot (visualised in figure 2.4). Optimality is usually defined with respect to the total transportation cost or distance, where the objective is to minimise this cost or distance, while respecting constraints such as vehicle capacities or customer time windows, or some other constraints depending on the variant of VRP. This problem generalises the famous TSP, which is the special case of VRP with a single vehicle visiting all customers, and hence the VRP inherits TSP’s computational complexity. In particular, the VRP is an NP-hard combinatorial optimization problem. As the number of customers increases, the number of possible route configurations grows exponentially, which means that the problem’s complexity blows up rapidly. For VRP instances beyond only a few locations, it becomes increasingly difficult for exact methods to provide an optimal solution within

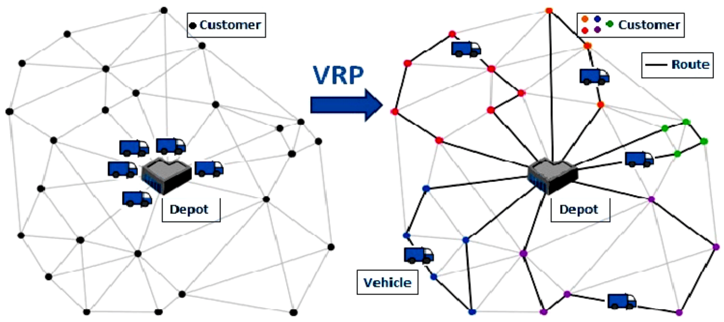


Figure 2.4: An arbitrary example instance of the classical Vehicle Routing Problem containing a single depot, multiple vehicles, and multiple customer nodes [25]

reasonably low computation times [27]. Optimal solutions can be found for small instances, but larger instances typically require advanced techniques or heuristics to produce good feasible solutions within a practical amount of time [28]. The VRP's complexity shows why combinatorial optimisation problems are so difficult to solve, because even a slight expansion of the input (more customers, vehicles, or added constraints) causes a combinatorial explosion in possibilities making brute force or other simple algorithms infeasible.

2.5. Multi-objective Optimisation

Multi-objective (combinatorial) optimisation problems (MOCO or MOCOPs) involve multiple objectives that need to be optimised simultaneously. These objectives are often conflicting, therefore, various solutions with a different trade-off among the objectives are considered. In a multi-objective combinatorial optimisation problem, the decision variables are discrete similar to COPs, but instead of one objective, each solution is evaluated on several objectives (i.e. two or more). For example, the goal of a routing problem could be to minimise the total travel time and total cost simultaneously. In a general sense, multi-objective (combinatorial) optimisation problems can be formulated as follows:

$$\min_{x \in X} F(x) = [F_1(x), F_2(x), ..., F_n(x)] \tag{2.10}$$

where $F(x)$ consists of n different objective functions, and $X \subseteq \mathbb{R}^d$ represents the search space. However, with these objectives typically conflicting one another, meaning by improving one objective the other may worsen, there is no single solution that is the best in all objectives simultaneously [29]. Therefore, in multi-objective optimisation the goal is to find a set of solutions, each considering a unique trade-off between the objectives. This set of solutions is commonly referred to as the non-dominated, or Pareto optimal solutions. Finding even a single one of these solutions can be NP-hard for many problems, and the number of Pareto optimal solutions could be exponentially large with respect to the size of the problem [6]–[9]. Thus, finding all the exact Pareto optimal solutions for a MOCO problem is extremely difficult and time consuming.

2.5.1. Pareto Optimal Solutions & Pareto Front

In multi-objective optimisation, a solution x is considered to be Pareto optimal if it cannot be improved with regards to any single objective without worsening at least one other objective [29]. Formally:

$$x \in X \text{ is Pareto optimal if and only if: } \nexists\, x' \in X \mid x' \neq x : \begin{array}{l} F_i(x') \leq F_i(x)\ \forall\, i \in \{1, 2, ..., n\} \wedge \\ \exists\, j \in \{1, 2, ..., n\} :\ F_j(x') < F_j(x) \end{array} \tag{2.11}$$

Equation 2.11 essentially states that a solution x is Pareto optimal if there is no other feasible solution x', that is different from x, and strictly better with respect to one objective and not worse in all other objectives. Since multiple trade-off solutions exist, the entire set of Pareto optimal solutions (one for each trade-off) is considered instead of only a single point. This of Pareto optimal solutions is called the Pareto set, and their respective mapping in the objective space is called the Pareto front. Figure 2.5 shows an arbitrary Pareto front that can result in a bi-objective optimisation problem. Each of the red points represent a Pareto optimal solution that considers a unique trade-off among some objectives F_1 and F_2. Together they form the Pareto front as highlighted in figure 2.5.

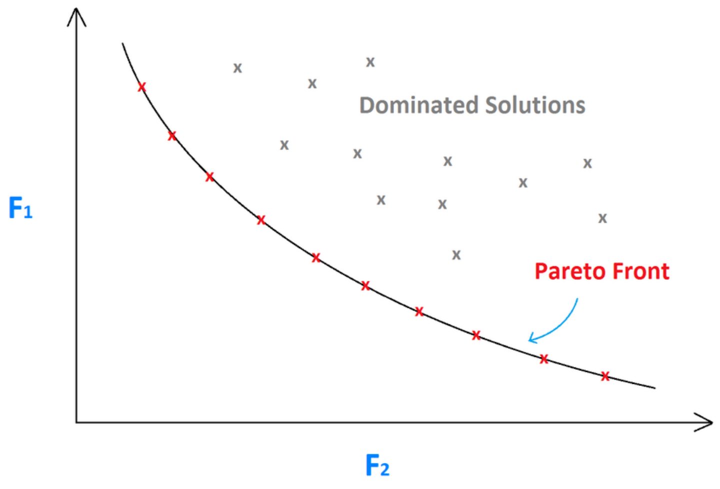


Figure 2.5: Pareto Front for an arbitrary bi-objective optimisation problem [30]

Decision-makers can examine the Pareto front to understand the trade-offs, because any point on this front cannot be dominated by any other, meaning it is an optimal solution for some preference of objectives. Typically, further decision analysis or preferences (e.g. assigning weights or using an interactive procedure) are needed to select a single final solution from the Pareto front according to the decision-maker's priorities. The Pareto optimality concept provides a framework for handling trade-offs, rather than collapsing multiple objectives into one, it identifies all the best trade-off solutions, providing support for informed decision-making in complex optimization problems [29].

2.6. Related Work

2.6.1. General MOCO Solutions

Solutions to MOCO problems can be divided into exact, approximation, and learning-based approaches. Exact methods are the most accurate, but unfortunately the most expensive approach. As mentioned before, exact approaches are often not feasible solutions due to their runtime. Especially, because most MOCO problems are NP-hard and the search space is extremely large with respect to the size of the problem [31]. This is why approximation ([32], [8]) and heuristic ([33], [34], [35]) based approaches were created to approximate the Pareto solutions or Pareto front within a reasonable time or cost limit. However, these approaches often require domain specific knowledge related to the problem to be optimised, because the operators used in these methods need to be designed specifically for each problem [6]. Often the level of knowledge and effort required to come up with these handcrafted designs is non-trivial [9].

2.6.2. Learning Based Approaches

To circumvent the need for such handcrafted designs and extensive domain knowledge, researchers have considered learning-based approaches. According to Bengio, Lodi, and Prouvost [36], the learning-based approaches for such problems can be characterised as *learning to configure algorithms* ([37], [38]), *learning alongside the algorithms* ([39], [40], [41]), and *learning to directly predict the solutions* ([42], [43], [44]). Combinatorial optimisation solutions using neural networks, also known as Neural Combinatorial Optimisation (NCO), falls under the latter. Vinyals, Fortunato, and Jaitly [45] designed a deep network, based on the sequence-to-sequence (seq2seq) neural architecture, to solve TSP instances using Supervised Learning. This model has had many improvements, including a crucial one by Bello, Pham, Le, *et al.* [46], which included RL such that the optimal solutions as labels were no longer needed. More recent works opted towards the use of Transformer networks using attention-based neural networks, rather than Recurrent Neural Networks (RNNs) using the seq2seq architecture, to solve TSP, CVRP, and other variants of VRP [47], [48].

2.6.3. AGH Related Solutions

However, none of these models focus on solving the global AGH problem, which is more complex than the classic VRPs due to the additional constraints. Only a few works explored the global optimisation of AGH instances. Among these works were several non-neural approaches, such as the decomposition based approach using constraint programming (CP) and large neighbourhood search (LNS) by Padrón, Guimarans, Ramos, *et al.* [3], and an evolutionary algorithm approach using the non-dominated sorting genetic algorithm (NSGA-II) by Liu, Wu, Tang, *et al.* [49]. Unfortunately, these approaches suffer from the same disadvantages of other meta-heuristic approaches as mentioned in section 2.6.1. Wu, Zhou, Xia, *et al.* [5] have introduced one of the earliest deep reinforcement learning based methods to solve AGH instances. Their solution provides a construction framework that decomposes the AGH problem into sub-problems for each fleet. All fleets are split into groups based on their precedence level, fleets with the same precedence level are placed in the same group. This ensures that the time windows of the flights being served in each level is not affected by other fleets in the same group. For each of the sub-problems solutions are constructed sequentially, and these solutions are used to update the time windows of flights for the succeeding precedence level. They use a DRL approach to learn construction policies to solve the sub-problems. Policy parametrisation is achieved by implementing an attention-based encoder-decoder model. The encoder learns representations of problem instances, and the decoder constructs solutions by learning which flight to select next for the current operation according to Wu, Zhou, Xia, *et al.* [5]. A complete overview of the policy network and its components is provided in figure 2.6.

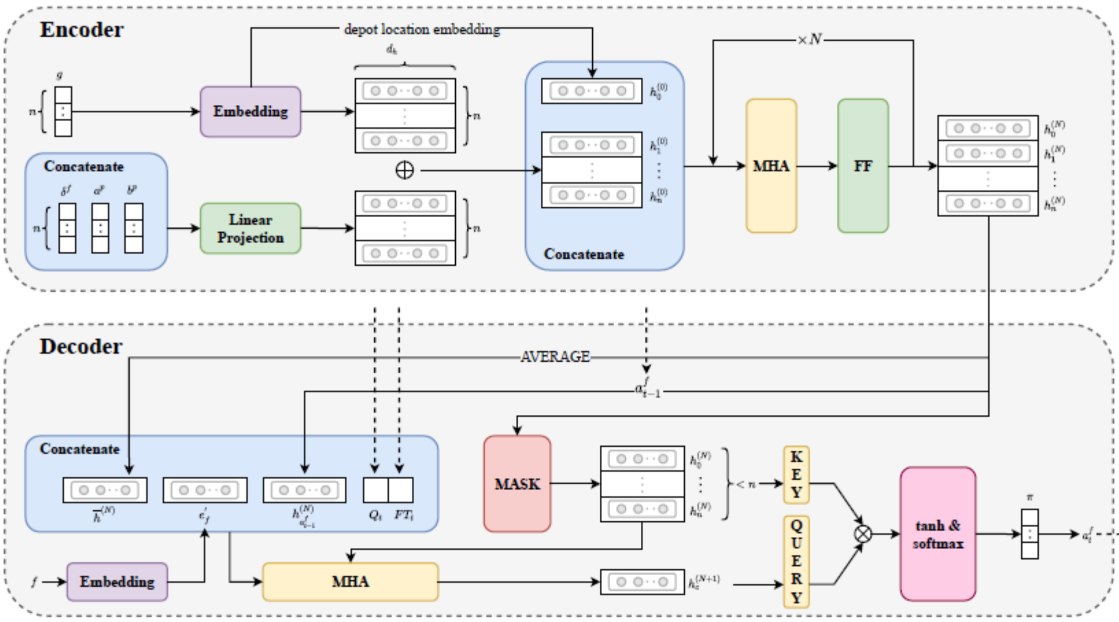


Figure 2.6: The complete policy network architecture [5]

The policy network is then trained by a RL algorithm called REINFORCE (introduced by Williams [50]) with a rollout baseline [16].

2.6.4. Multi-Objective AGH Solutions

Although, the model implemented by Wu, Zhou, Xia, *et al.* [5] considers the global optimisation of AGH, they focus solely on single-objective optimisation of the AGH problem. The only attempt that has been made towards solving multi-objective AGH instances is by [3]. However, they have done so by using an Evolutionary Algorithm approach. Such meta-heuristic approaches have the disadvantage of requiring extensive domain specific knowledge for any given problem, and the solution to any problem does not generalise well as it is tailored specifically towards a single problem. To the best of our knowledge, there have been no other attempts that try to solve the global optimisation of the multi-objective AGH problem.

Although, no specific Neural Combinatorial Optimisation solutions have been proposed to solving the multi-objective AGH problem, researchers have come up with solutions targeting multi-objective optimisation of general VRPs, such as TSP or CVRP. Among these approaches are the models by Li, Zhang, and Wang [51], Wu, Wang, and Zhang [52], Zhang, Wang, Zhang, *et al.* [53], and Zhang, Wu, Zhang, *et al.* [54]. All of these solutions use the decomposition based multi-objective evolutionary algorithm (MOEA/D) framework [34], to decompose a MOCO problem into numerous single-objective sub-problems. The solutions then build a model for each sub-problem, and solve each sub-problem separately [9].

However, since the number of Pareto solutions would be exponentially large with respect to the input size, the required number of models would be huge for finding the whole Pareto set according to Lin, Yang, and Zhang [9]. Besides this issue, the researchers or consumers of their models cannot dictate which Pareto optimal solutions will be selected for the approximation of the Pareto front, i.e. they have no control over the approximation of the Pareto front as they cannot determine which trade-off points the models should find. Therefore, Lin, Yang, and Zhang [9] have suggested a novel approach in which a single model is used to solve MOCO problems. The architecture of their proposed model is displayed in figure 2.7.

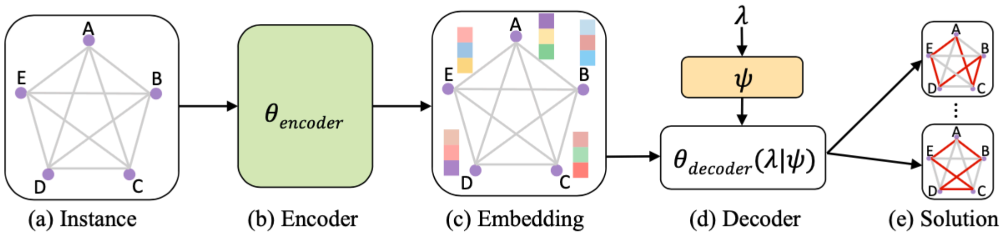


Figure 2.7: The architecture of the Preference-conditioned Neural MOCO Model [9]

As can be seen in figure 2.7, their model uses an encoder-decoder structure, but it is slightly modified to allow for more control of the resulting Pareto optimal solution. They have implemented an Attention Model similar to that of Kool, Hoof, and Welling [48], but they have incorporated the use of preference vectors to indicate the trade-off or priority between the objectives. The shared attention encoder is given some input, which is an instance of a MOCO problem, and it outputs a set of embeddings. These embeddings are not dependent on the preference vector, i.e. for any given instance the same embeddings can be used for various preferences. However, they have slightly adjusted the decoder implemented by Kool, Hoof, and Welling [48], by conditioning the decoder parameters on some preference vector which they denote by λ. To generate these preference-conditioned parameters they use a multi-layer perceptron (MLP) hypernetwork, formally $\theta_{decoder} = MLP(\lambda|\psi)$. The decoder takes as input the node embeddings, which are outputted by the encoder and some sampled preference vector, and it outputs a feasible solution to the given problem. For example, for multi-objective TSP (or MOTSP), it would output a feasible tour between the cities in the given problem instance [9].

2.6.5. Research Gap

The related work, discussed in the foregoing sections, answers the second research questions posed in Chapter 1 section 1.3. There is only one approach that solves the complete AGH problem for multiple objectives, which is the approach by Padrón, Guimarans, Ramos, *et al.* [3]. However, it applies a meta-heuristic approach, such approaches have the disadvantage of requiring manually engineered operators that exploit the researchers' domain specific knowledge, and they often generalise poorly to other types of instances. Thus, current research does not apply neural solutions to the multi-objective AGH problem with all the constraints (capacity, precedence relations, and time windows), and while one neural method has been proposed for the complete AGH problem, it is restricted to single-objective optimisation. Other approaches, both neural and other types, either relax constraints, omit parts of the AGH problem, or focus on individual operations, thereby failing to address the AGH problem in its entirety. Meanwhile, neural combinatorial optimization methods for multi-objective variants of routing problems such as the TSP or CVRP show promise, but fall short in two critical ways. Firstly, they often require training multiple models to approximate the Pareto set, which limits scalability as the number of required models would be huge with respect to the input size [9]. Secondly, they provide no control over which Pareto optimal solutions are approximated. However, Lin, Yang, and Zhang [9] have proposed a neural approach using a preference-conditioned, attention-based encoder-decoder network, which has shown promising results on several multi-objective routing problems; but it has not been applied to the multi-objective AGH problem. Altogether, this highlights a gap in the current literature, which is that no existing approach leverages a neural model that can learn to optimise the complete multi-objective AGH problem without requiring (extensive) domain specific knowledge, and while still offering control over the Pareto optimal solutions that will be considered. Given the promising results of the model proposed by Lin, Yang, and Zhang [9], and its similarity in network structure to the DRL model solving single-objective AGH, we seek to address this gap in the literature by combining the techniques of these approaches to come up with a neural model that is able to learn to optimise the multi-objective AGH problem.

3 Methodology

Given the gap in the current literature, as identified in Chapter 2, we intend to contribute by proposing a novel deep reinforcement learning (DRL) solution that uses a single model, which can generate the Pareto set (or its equivalent Pareto front) using sampled preference vectors, for multi-objective optimisation of the airport ground handling (AGH) problem. Several variants of the model have been implemented as part of this research to explore potential improvements to the initial model that was created. The initial model that was implemented will be referred to as the baseline model hereafter. To understand the baseline model and its variants, this chapter starts with an overview of the approach used in this research in section 3.1, which explains the process of extending existing work to support optimisation of multi-objective airport ground handling. This is followed by section 3.2, in which the implementations of the baseline model and the hypothetical improvements are explained. The code for this project can found on https://github.com/Mozafar7/MO-AGH.

3.1. Approach Overview

As explained before in sections 2.6.1 and 2.6.4, meta-heuristic approaches require extensive domain knowledge to come up with an efficient model, they lack generalisation to other types of instances, and are not efficient on larger instance sizes, making these approaches less appealing than neural approaches. In contrast, neural approaches do not require the domain knowledge, are able to generalise well to varying instances, and have the potential to efficiently solve larger instances. Especially, deep reinforcement learning models, which do not require truth labels as is the case in Supervised Learning. That is why we have chosen to pursue a DRL approach to solving the multi-objective AGH problem. The only neural approach towards solving the global AGH problem, is a DRL model, implemented by Wu, Zhou, Xia, *et al.* [5], which has shown promising results for single-objective optimisation of the AGH problem. However, since we are focussed on attempting to solve the global multi-objective AGH problem, our goal is to extend the existing model such that it can solve multi-objective AGH instances.

Since no existing solutions leverage the neural models' ability to learn from problem instances without prior domain knowledge, and its ability to generalise, on the multi-objective AGH problem, we explored neural approaches applied to other MOCO problems as potential candidates for extending the single-objective model. Among these, we focus on the approach proposed by Lin, Yang, and Zhang [9], which employs a similar encoder-decoder architecture as the single-objective DRL model of Wu, Zhou, Xia, *et al.* [5]. Despite some differences, this structural similarity should ease the incorporation of its preference-conditioned multi-objective logic into the existing framework of the single-objective DRL model. Moreover, the approach of Lin, Yang, and Zhang [9] is able to learn (approximations of) Pareto fronts of MOCO problems with a single model, and it provides explicit control over the trade-off points that are considered for the Pareto front, thereby overcoming limitations of alternative approaches. Other approaches, as discussed in Chapter 2, generally require a set of models to solve a MOCO problem, which could become exponentially large with respect to the input size, and do not provide any control with regards to the Pareto optimal solutions generated. Therefore, the approach proposed by Lin, Yang, and Zhang [9] makes for a strong candidate for extending the single-objective DRL model for

AGH to support multi-objective optimisation.

The architecture of both these models is inspired by the attention model of Kool, Hoof, and Welling [48]. Hence, both models consist of an encoder that computes embeddings given some problem instance as its input, and a decoder that is given the embeddings of the encoder as its input and it creates solutions (or some stochastic policy to generate a solution). The major difference between them being that the decoder of Lin, Yang, and Zhang [9] is preference-based, meaning that its parameters are generated by a MLP hypernetwork that takes the 2 dimensional preference vector (in the case of 2 objectives) as its input and generates the parameters for the decoder. This decoder then generates an approximate Pareto solution taking the trade-off defined in the preference vector into account. The high-level architecture of the models are displayed in figures 2.3 and 2.7, respectively. On the other hand, the encoders of both models are preference agnostic, i.e. the generated node embeddings do not depend on any preference between the objectives. Therefore, the generated embeddings can be used across all preference vectors that will be considered. Also, the encoder logic of the single-objective model does not have to change. Thus, we focussed on extracting the preference-conditioned logic and incorporating it into the single-objective model for AGH. The initial model that combines the logic of the two models will be referred to as the baseline model.

Learning in RL algorithms involves two main steps, policy evaluation, i.e. estimating the value of states and actions given some policy, and policy improvement, i.e. updating the current policy to prefer actions that yield higher value rewards. Optimisation algorithms, such as value iteration, policy gradient, Q-learning, and so on, all rely on maximising or minimising a scalar objective. Therefore, a vector reward would require some predefined scalarisation, such as a weighted sum, to combine the multiple values in the vector into a scalar reward; otherwise the Bellman Optimality Equations (as defined in Chapter 2) and convergence guarantees would break down [16], [55]. Since the model now has two objective values, they must be combined into a single scalar value for it to be compatible with the RL algorithm. For the baseline model this combination has been realised by applying a weighted linear combination of the objective values, with as weights the values of the corresponding sampled preference vector, because this is a relatively simple and intuitive way to apply scalarisation. Also, it allows for direct encoding of the relative priorities of the objectives by using the preference vector as weights. Formally, the reward, as a result of the weighted sum, can be described as:

$r = w_1 \cdot F_1 + w_2 \cdot F_2$, where w_1 and w_2 represent the preference of each objective value F_1 and F_2 (3.1)

The preference vector is a 2-dimensional vector, with each value in $[0, 1]$, and the sum of the values adding up to 1. As hypothetical improvements we have considered applying Tchebycheff scalarisation (TCH scalarisation) to map the objective values into a single scalar for the reward, rather than the weighted linear combination. The reason why we consider this scalarisation method is explained in hypothesis 1. Formally, this scalarisation is defined as:

$$\min_{x \in X} g(x) = \max_{1 \leq i \leq m} w_i |f_i(x) - z_i^*| \tag{3.2}$$

where:

- $f_i(x)$ is the value of objective i
- z_i^* is some ideal point (i.e. the best possible value in each objective)
- $w_i > 0$ are weights (the preferences in this case)
- g(x) is the scalarised function (i.e. the reward value)

TCH minimizes the worst-case weighted deviation from some ideal point [55], [56].

Additionally, we considered using latent vectors instead of the full matrices in the current model, this model will be referred to as the latent vector model hereafter. The reason why we think that this could be beneficial is explained in hypothesis 2. Finally, we considered a combination of both the baseline model with TCH scalarisation, and the latent vector model with TCH scalarisation, to verify whether a combination of these suggested improvements can lead to even better results. So, we considered four variants of the model, the baseline model, the baseline with TCH scalarisation, the latent vector model with a weighted sum combination of the objectives, and the latent vector model with TCH scalarisation. Our hypotheses for each addition are as follows:

1. TCH scalarisation, unlike weighted sums, can generate Pareto solutions in both convex and non-convex regions of the Pareto front [55], [56]. So, it should allow us to more accurately and consistently produce any type of Pareto front regardless of its shape, but probably at the cost of computational cost as this requires a somewhat more complex calculation than a weighted sum.

2. Using latent vectors instead of full matrices will reduce the complexity of computations as the dimensionality will be reduced [19], resulting in more efficient training times, which will allow for more training within the same time limits or training for a similar number of epochs with reduced costs.

The validations of these hypotheses will be based on the results of the experiments that were conducted. These are discussed in Chapter 4.

3.2. Implementation

As mentioned before, the model now has to have two objectives in order to utilise the 2-dimensional preference vectors. Since the single-objective model calculates only a single objective value, namely the total travel distance, we must determine the second objective value. The second objective being the latest completion time, as mentioned in Chapter 1, we decided to leverage the existing logic of the single-objective model to calculate the score related to this objective. The current time of any fleet vehicle is determined by its starting time and its duration. The duration in turn is determined by the distance it travelled divided by the vehicle speeds defined in the existing model. Whenever a vehicle goes back to the depot node it receives a negative time reset in the existing logic, however the experiments in Chapter 4 show that this can be troublesome in the multi-objective setting we intend to explore. The two objective values must be scalarised, i.e. the two values must be combined into a single scalar, because the REINFORCE algorithm used for policy training expects a scalar reward value. We consider two approaches to the scalarisation, a weighted sum approach, and a weighted TCH scalarisation, as mentioned in section 3.1. Given that we do not have the truth labels or optimal values for the AGH instances, we use the minimum values encountered as the optimal values in the equation for TCH scalarisation (see equation 3.2).

To enable the decoder to learn preference-based solutions, the single-objective model should include the hypernetwork, that generates the weights for the decoder, such that it becomes preference conditioned. By extracting the hypernetwork that generates the parameters for the decoder based on the preference vector, we can keep the decoder of the single-objective AGH model as is. Also, other logic of the existing single-objective AGH model can remain the same, only the parameters of the decoder would be generated by this added hypernetwork. For the structure of the existing single-objective model, please refer to the original paper by Wu, Zhou, Xia, *et al.* [5]. The hypernetwork is essentially a multi-layer perceptron (MLP) with two hidden layers of dimensionality 128, and it applies a ReLu activation function [57]. The input is a 2-dimensional preference vector, with each value in $[0, 1]$, and the sum of the values adding up to 1. Formally, $\lambda_i \in [0, 1]$ and $\sum_{i=1}^{2} \lambda_i = 1$, where λ_i denotes the preference for the i-th objective. The MLP model first produces a hidden embedding, denoted as $e(\lambda) = MLP(\lambda|\psi)$, which is subsequently mapped to the decoder parameters through a linear projection. This linear projection is formally defined as $\theta_{decoder} = We(\lambda) + b$ [9].

By incorporating the hypernetwork logic into the single-objective model, the parameters of the decoder in the single-objective model changes. Formally, this change can be represented as:

$$\theta = [W_Q, W_K, W_V, W_{MHA}] \rightarrow \theta(\lambda) = [W_Q(\lambda), W_K(\lambda), W_V(\lambda), W_{MHA}(\lambda)] \tag{3.3}$$

Where the original parameters are represented to the left of the arrow, and the parameters generated by the hypernetwork are represented by the equation on the right of the arrow. Essentially, what that means is that the parameters are now dependent on some preference vector λ. Since these parameters will become preference conditioned, consequently turning the MHA mechanism into a preference based MHA and also the calculation of the score of selecting each action becomes preference based. The score calculation is defined in [5] as:

$$v_t(j) = \begin{cases} C \cdot \tanh(\frac{(W_Q \mathbf{h}_c^{(N+1)})^T (W_K \mathbf{h}_j^{(N)})}{\sqrt{d_h}}), & m_t^j = \text{FALSE} \\ -\infty, & m_t^j = \text{TRUE} \end{cases} \tag{3.4}$$

The score calculation will now become:

$$v_t(j,\lambda) = \begin{cases} C \cdot \tanh(\frac{(W_Q(\lambda) \mathbf{h}_c^{(N+1)})^T (W_K(\lambda) \mathbf{h}_j^{(N)})}{\sqrt{d_h}}), & m_t^j = \text{FALSE} \\ -\infty, & m_t^j = \text{TRUE} \end{cases} \tag{3.5}$$

where m_t^j denotes the masking of flight j at the current time step t, when it is true it means that the flight cannot be selected at the current step.

The probability distribution over the candidate actions depends on the score calculation. Hence, the probability distribution over candidate actions now also depends on the preferences. The probability distribution changes, from the original formula defined in equation 3.6 [5] to the updated formula defined in equation 3.7.

$$\pi_\theta(a_t^f | x^f, a_{1:t-1}^f) = \frac{e^{v_t(a_t^f)}}{\sum_{j=0}^{n} e^{v_t(j)}} \tag{3.6}$$

$$\pi_{\theta(\lambda)}(a_t^f | x^f, a_{1:t-1}^f) = \frac{e^{v_t(a_t^f,\lambda)}}{\sum_{j=0}^{n} e^{v_t(j,\lambda)}} \tag{3.7}$$

Accordingly, the stochastic policy expressed by the encoder-decoder neural network becomes preference conditioned, changing the policy formula from equation 3.8 [5] to equation 3.9. The policy generates a solution (tour) a^f for the fleet (operation) f, given instance x^f [5].

$$\pi_\theta(a^f | x^f) = \prod_{t=1}^{T} \pi_\theta(a_t^f | x^f, a_{1:t-1}^f) \tag{3.8}$$

$$\pi_{\theta(\lambda)}(a^f | x^f) = \prod_{t=1}^{T} \pi_{\theta(\lambda)}(a_t^f | x^f, a_{1:t-1}^f) \tag{3.9}$$

where T represents the number of steps used to construct a feasible tour.

An overview of the architectural change is given in figure 3.1. The architecture is an extension of the architecture overview of the single-objective AGH model, which was provided in figure 2.3. The decoder now has an additional input, which is the output of the hypernetwork, i.e. the parameters generated by the hypernetwork. The hypernetwork has a single input, which is the m dimensional preference vector λ, where m denotes the number of objectives. In our case $m = 2$, as we are optimising for two objectives.

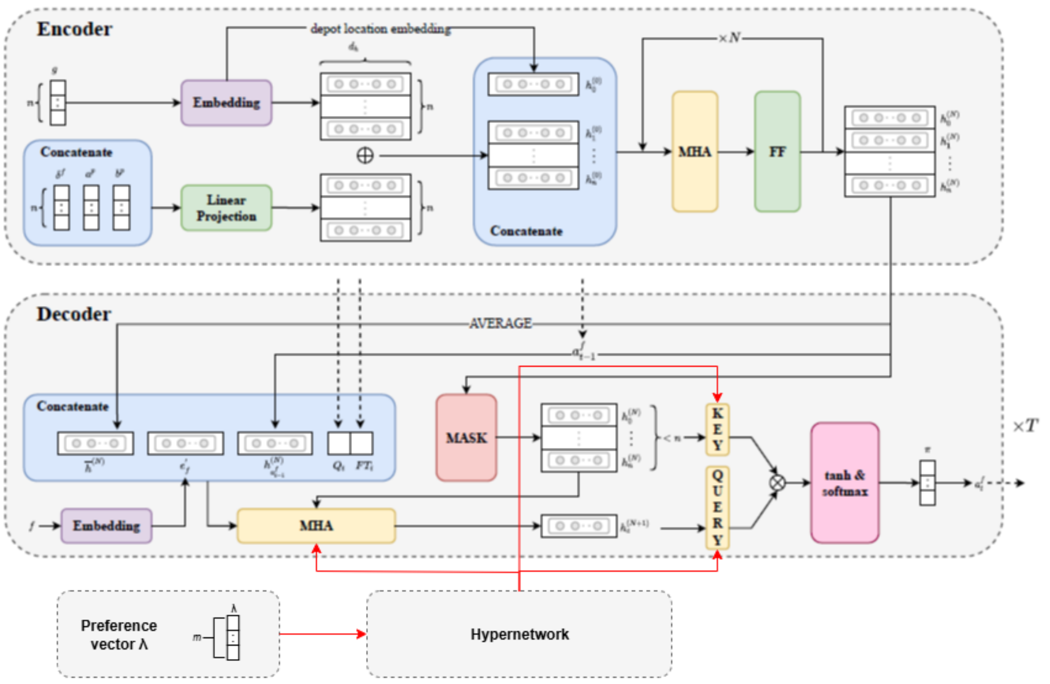


Figure 3.1: A high-level overview of the extended architecture, which incorporates the preference-based hypernetwork into the single-objective DRL model architecture

3.2.1. Latent Vector Approach

An additional hypothetical improvement (see hypothesis 2 in section 3.1) we have implemented is using latent vectors rather than full weight matrices to reduce the computational complexity of the hypernetwork. In the original logic the hypernetwork outputs the entire weight matrices for the parameters of the decoder (W_Q, W_K, W_V, W_{MHA}). In the latent vector approach we intent to structure the hypernetwork such that it outputs a relatively small latent vector per projection group, which is then passed through the dedicated learned expansion layers (one per weight matrix) to approximate the full weight matrices. The disadvantage of this approach would be that it is slightly more complex to implement than the full weight matrices as it will contain more components, and it is likely less expressive than the full weight matrices because we have less weights to express the underlying problem. However, we expect the latent vector models to train more efficiently as it significantly reduces the number of parameters and thereby the complexity of the calculations.

Each of the approaches mentioned above has been trained on AGH instances of size 20, 50, 100, and 200 nodes. These instances were generated and included in the code of the single-objective DRL model [5]. The experiments that were conducted, and their respective results, are discussed in Chapter 4.

3.2.2. Evolutionary Algorithms

Besides comparing the results of the implemented models against each other on the respective instance sizes, and on how well they generalise to other instance sizes, we have also implemented two evolutionary algorithms to compare their performance to the neural models on the same instances. Firstly, we have implemented the Non-Dominated Sorting Genetic Algorithm II, introduced by Deb, Pratap, Agarwal, *et al.* [58], as it has become a standard meta-heuristic approach for solving multi-objective optimisation problems because of its efficiency and effectiveness in approximating the Pareto front [58], [59]. Secondly, we have implemented the Decomposition-based Multi-Objective Evolutionary Algorithm (MOEA/D), introduced by Zhang and Li [34], which decomposes multi-objective problems into sub-problems that are solved separately. Instead of comparing against all individuals via Pareto dominance, as in NSGA-II, MOEA/D exploits neighbourhood structures of the sub-problems, which should make it more efficient on large populations [34]. We chose MOEA/D because of its potential for better scalability than NSGA-II, and because it has also become a standard in evolutionary approaches to multi-objective optimisation.

As the primary aim of this research was not the development of novel state-of-the-art meta-heuristic approaches, we have used the pymoo framework [60], which allows you to define a single problem definition and then apply several solving algorithms to that problem definition. For the pymoo problem definition we have converted the extended MILP problem definition, defined in Chapter 1 section 1.1, into a pymoo problem class. The parameters for the evolutionary algorithms were tuned empirically, using iterative testing to identify value that achieved the best observed performance with regards to the objective scores and the spread along the respective Pareto fronts. Listings 3.1 and 3.2 show the configurations of the evolutionary algorithms NSGA-II and MOEA/D, respectively. In the experimental setup the number of generations ('n_gen' in code) of both algorithms is dynamically adapted to the increasing instance sizes, i.e. for larger instance sizes we use a larger number of generations, because of the increased complexity and increased search space. The results of the experimental evaluation of these algorithms, and the neural models, are discussed in section 4.3.

```
def run_nsga2(problem, seed, n_gen, verbose=False):
    algorithm = NSGA2(
        pop_size = 100,
        sampling = LHS(),
        crossover = SBX(prob=0.7, eta=15),
        mutation = PM(prob=0.7, eta=5),
        eliminate_duplicates = False,
    )

    result = minimize(
        problem,
        algorithm,
        ('n_gen', n_gen),
        seed=seed,
        verbose=verbose
    )

    return result
```

Listing 3.1: NSGA-II Algorithm Parameters

```
def run_moead(problem, seed, n_gen, num_points, verbose=False):
    ref_dirs = get_reference_directions("energy", 2, n_points = num_points, seed=seed)
    algorithm = MOEAD(
        ref_dirs = ref_dirs,
        n_neighbors = 3,
        prob_neighbor_mating = 0.9,
        crossover = SBX(prob=0.9, eta=15),
        mutation = PM(prob=0.8, eta=5),
        decomposition = Tchebicheff(),
    )

    result = minimize(
        problem,
        algorithm,
        ('n_gen', n_gen),
        seed=seed,
        verbose=verbose
    )

    return result
```

Listing 3.2: MOEA/D Algorithm Parameters

4 Experiments

This chapter presents the results of the conducted experiments of the proposed approaches. The experiments compare the baseline model, which is the initial combination of the single-objective AGH model defined by Wu, Zhou, Xia, *et al.* [5] and the Pareto Set Learning approach [9], several hypothetical improvements, and two Evolutionary Algorithms (NSGA-II & MOEA/D) approaches.

First a comparison is made with regards to the training results of the naive combination of the models mentioned in Wu, Zhou, Xia, *et al.* [5] and Lin, Yang, and Zhang [9], and the hypothetical improvements (TCH scalarisation, latent vector model, etc.). Afterwards, an analysis of the models' ability to generalise to unseen instances, of sizes different to the ones they were trained on, will be discussed. This is followed by a comparison between the performance of the trained models and two EA approaches. All of the experiments were run on the DelftBlue Supercomputer using the resources that were made available for master students. The experiments were run on the GPU-A100 node of DelftBlue, using 2 CPUs per task, and 6 GB memory per CPU [61].

4.1. Training

The various models that were implemented during this project, as mentioned in Chapter 3, have been trained on varying instance sizes, from smaller instances of 20 nodes up to larger instances of 200 nodes. For all instances the training data for a single epoch was divided into 4 batches. All the models were trained for 100 epochs or a maximum time limit of 24 hours. This time limit was determined by Delft Blue for the resources that this project was entitled to.

The resulting graphs, in sections 4.1.1, 4.1.2, 4.1.3, and 4.1.4, show the performance of 4 models that have been implemented. The performance is shown with regards to the instance sizes that the models were trained on, and the two objectives that should be minimised in the multi-objective AGH problem defined in Chapter 1. These two objectives are the total travel distance of all fleet vehicles and the latest completion time of any task in the generated schedule. During every epoch the input was divided into 4 batches, so the average and the variance over these 4 batches is shown per epoch in the graphs. Followed by statistics with regards to the training efficiency of the implemented models for the various instance sizes in section 4.1.5.

4.1.1. Results 20 nodes

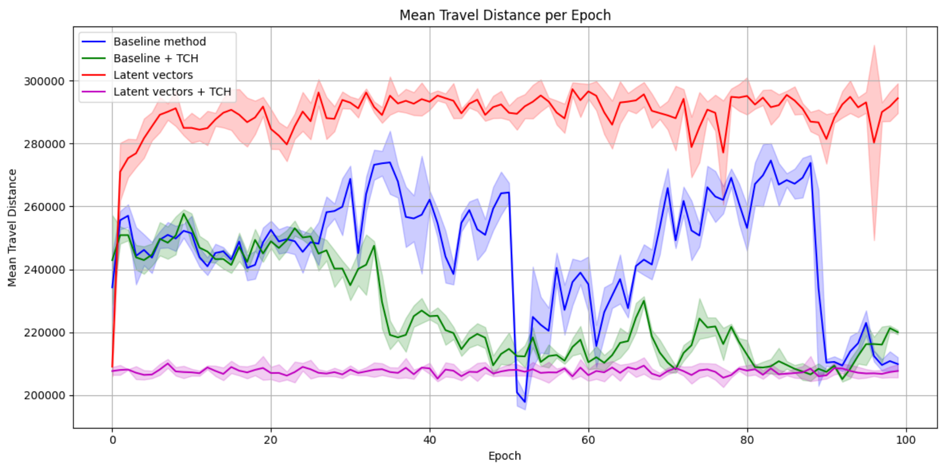


Figure 4.1: The average total travel distance per epoch for instances with 20 nodes

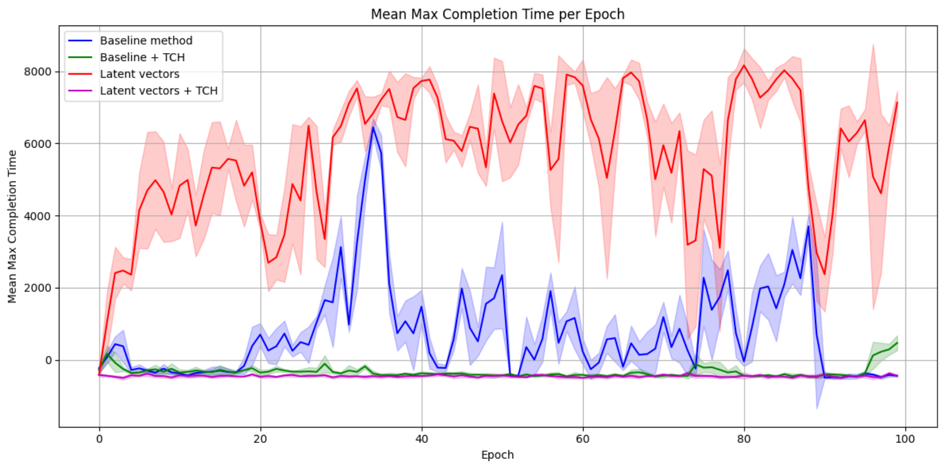


Figure 4.2: The average maximum completion time (makespan) per epoch for instances with 20 nodes

What can be concluded from graphs 4.1 and 4.2 is that the baseline model and the latent vector model perform poorly on both objectives in comparison to the models where Tchebycheff (TCH) scalarisation is applied. Also, for both those models the variance within an epoch for the completion time objective are significantly larger than the other models, even in later epochs. Although, the baseline model seems to converge to a similar objective scores as the other models, there is a significantly larger variance in the performance, both within a single epoch and in between epochs. With the application of Tchebycheff scalarisation the performance of the models seems to smoothen somewhat, less variance is noticed within epochs and the graphs contain fewer steep changes. Thus, based on these graphs, applying Tchebycheff scalarisation stabilises the performance of the models while ensuring convergence to similar, if not better, objective scores.

4.1.2. Results 50 nodes

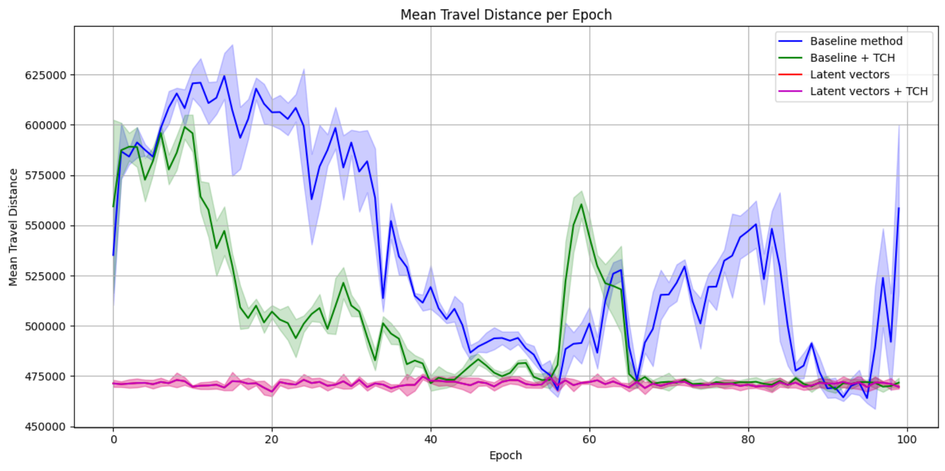


Figure 4.3: The average total travel distance per epoch for instances with 50 nodes

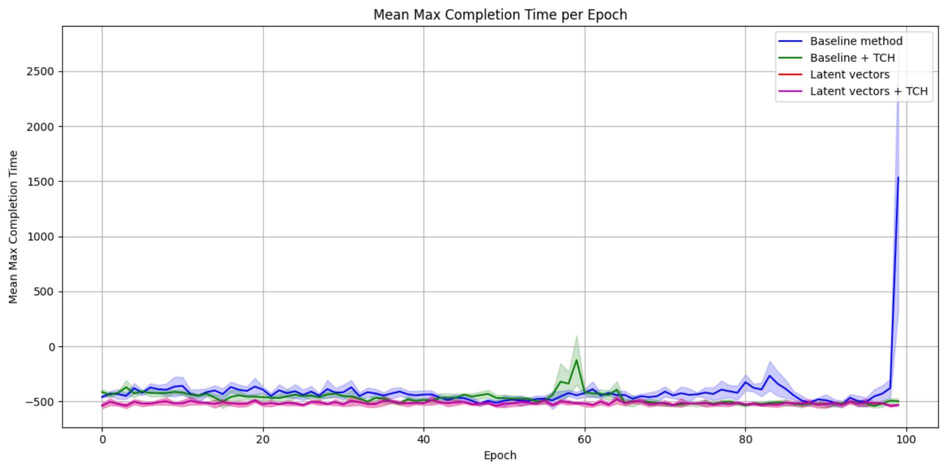


Figure 4.4: The average maximum completion time (makespan) per epoch for instances with 50 nodes

For instance sizes of 50 nodes (graphs 4.3 and 4.4) the latent vector models perform exactly the same, with 10 runs yielding in the exact same results for both variants of the latent vector model. However, what is unexpected is that these models have barely any variance with regards to both objective scores, especially the travel distance. In contrast, the baseline models seem to perform as expected, there is some variance in the objective scores, starting from poor results and slowly getting better. However, only the baseline model with TCH scalarisation seems to converge well for both objectives, but even this model has a large spike around epoch 60, but then converges rapidly. Whereas, the baseline model without TCH scalarisation converges later and is not stable enough to remain around these objective scores. Notice the spikes around the final epochs for the naive baseline model with regards to both objective scores. Thus, based on the graphs in figures 4.3 and 4.4, the baseline model with TCH scalarisation has the most promising learning curves with respect to both objectives, because it highlights its ability to learn and perform better as the epochs progress.

4.1.3. Results 100 nodes

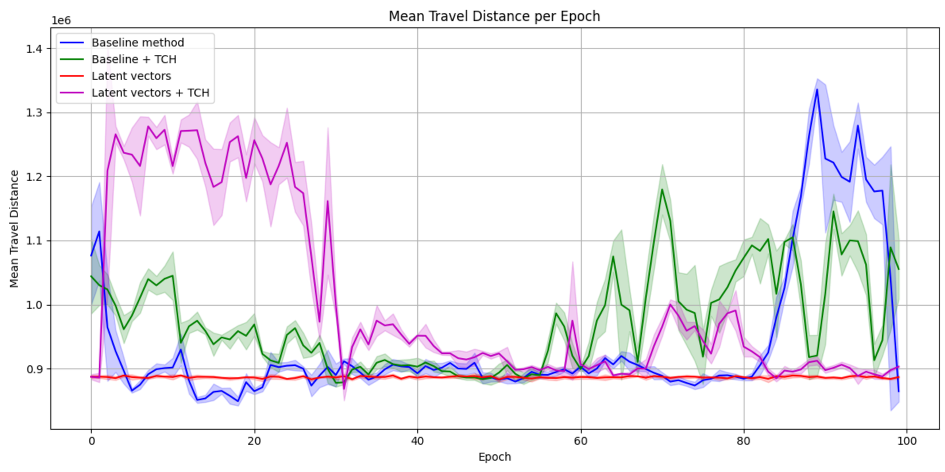


Figure 4.5: The average total travel distance per epoch for instances with 100 nodes

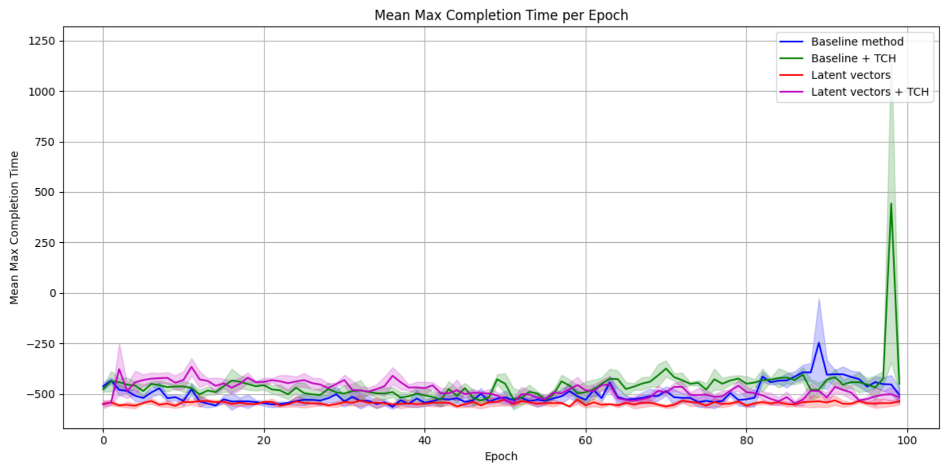


Figure 4.6: The average maximum completion time (makespan) per epoch for instances with 100 nodes

In contrast to the latent vector models' performance for 20 nodes (figure 4.1), the TCH variant seems to have more variance than the non-TCH version during training on 100 nodes (figure 4.5). Where the latent vector model with TCH scalarisation has a very stable performance on both objectives during training on 20 nodes, here it is quite the opposite with very large variances between epochs and also significantly larger variances within an epoch. The baseline method on the other hand is able to converge with very little variance within an epoch and between epochs up to epoch 80, but then suddenly spikes up significantly and converges back to the value that was encountered in epoch 80. However, the TCH variant of the baseline also has a less stable performance on both objectives compared to the non-TCH baseline variant, contrary to the training data for 20 (figure 4.1) and 50 nodes (figure 4.3). For the maximum completion time during training on 100 nodes, the scores of all models are dominated by the negative reset when the depot node is visited, although the baseline with TCH scalarisation does have a peak around epoch 98 (figure 4.6). Based on the training curves with respect to the total travel distance objective, which can be seen in figure 4.5, the latent vector models have the most promising learning curves; because they are able to converge well and to lower objective scores than the other models.

4.1.4. Results 200 nodes

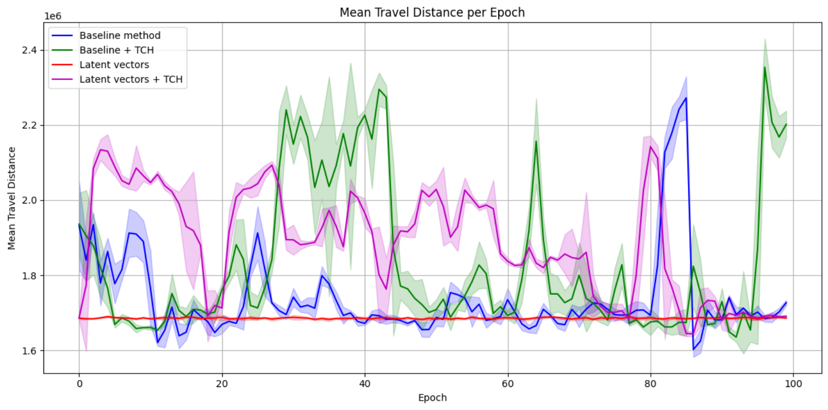


Figure 4.7: The average total travel distance per epoch for instances with 200 nodes

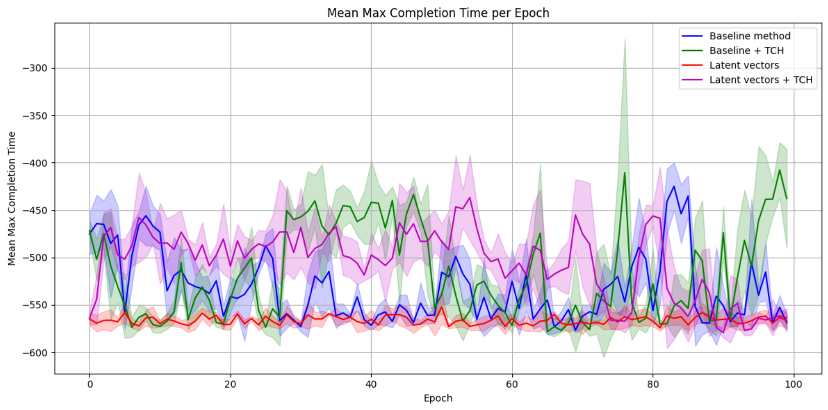


Figure 4.8: The average maximum completion time (makespan) per epoch for instances with 200 nodes

Similar to the models in 100 nodes, but contrary to the models in the smaller instances, the TCH variants of the models for 200 nodes also have larger fluctuations in the objective values during training. Again the latent vector model with TCH scalarisation has more variance than baseline version (figure 4.7), quite similar to what can be seen during the training on 100 nodes (figure 4.5). Although convergence occurred in earlier epochs during training on 100 nodes, which happened around epoch 50 for 100 nodes, for the latent vector model with TCH scalarisation convergence occurs around epoch 85 on 200 nodes. The baseline method on 200 nodes has a very analogous pattern to that of the baseline method on 100 nodes. It too is able converge with very little variance, both within an epoch and between epochs, up to epoch 80. Same as the baseline method on 100 nodes, it has a sudden spike after epoch 80 and then rapidly converges again. However, the TCH variants of the baseline of 100 nodes and 200 nodes do not perform as much alike. Where the baseline model with TCH scalarisation is able to converge up to around epoch 55 on 100 nodes (figure 4.5), and then starts to lose its ability to converge to a low objective value on average, the baseline model with TCH scalarisation on 200 nodes is generally not able to converge properly to a low mean objective score. This behaviour of the baseline model with TCH scalarisation on 200 nodes is not in line with its behaviour on 20 nodes (figure 4.1), 50 nodes (figure 4.3), or 100 nodes (figure 4.5). For the maximum completion time during training on 200 nodes, the scores of all models are dominated once again by the negative reset when the depot node is visited (figure 4.8). Based on the learning curves of the models in figures 4.7 and 4.8, the baseline method has the most potential for finding the best solutions with regards to both objectives, because it is able to learn and converge over the 100 epochs with the least amount of variance throughout the training epochs.

4.1.5. Training Efficiency

Besides the models being able to perform well with respect to the objectives, they must also be able to train within reasonable time. To compare the models, their training times per epoch were tracked and aggregated over the 100 epochs that they were trained on. Table 4.1 shows these average training times over the 100 epochs per model and instance size, with their respective standard deviations noted within the parentheses. Based on the resulting data, the latent vector models are able to train faster compared to their baseline counterparts. The latent vector model takes 10.48% less time on average per epoch, and the TCH variant takes 4.51% less time on average per epoch compared to the respective baseline models. This means that the latent vector models can be trained for more epochs within the same time. Depending on how these models perform on the evaluations, which are discussed in section 4.3, it means that the latent vector models can be trained either to perform as good as the baseline models in less time, or they can be trained for the same amount of time while reducing the computational costs needed per epoch. Additionally, there is a significant reduction in the variances of the training times of the latent vector models in comparison to the baseline models. This indicates that the training times of the latent vector models is much more stable than the baseline variants, confirming the latent vector hypothesis made in Chapter 3 section 3.1. Besides confirming the hypothesis that the latent vector models will result in more efficient training times, it also shows that these models result in more stable training times given the significantly lower standard deviation than their baseline counterparts.

Instance Size	Baseline	Baseline + TCH	Latent Vector	Latent Vector + TCH
20	160.64s (4.94s)	143.66s (6.67s)	149.99s (2.02s)	130.63s (1.34s)
50	332.03s (24.43s)	296.81s (26.05s)	283.13s (2.92s)	275.12s (1.82s)
100	550.17s (58.68s)	596.12s (57.51s)	508.84s (3.37s)	599.87s (6.82s)
200	1090.69s (72.77s)	1113.22s (133.49s)	991.37s (25.49s)	1101.86s (72.42s)

Table 4.1: Average training time per model and instance size (with standard deviation inside the parentheses)

4.2. Scalability & Generalisation

This section highlights the scalability and generalisation of the implemented models by looking at how well the models, trained on varying instance sizes, perform on unseen instances of AGH with a different size. It focusses on the instance sizes 20, 50, and 100 nodes. For each of the instance sizes the models trained on the respective instance size and models trained on the two other sizes are compared against each other. This comparison is done by looking at the Pareto fronts generated by each of the models to see which one is able to generate a Pareto front that minimises the objectives. The results are shown for the instance sizes 20, 50, and 100 nodes, in the subsections 4.2.1, 4.2.2, and 4.2.3, respectively.

4.2.1. Instances of 20 nodes

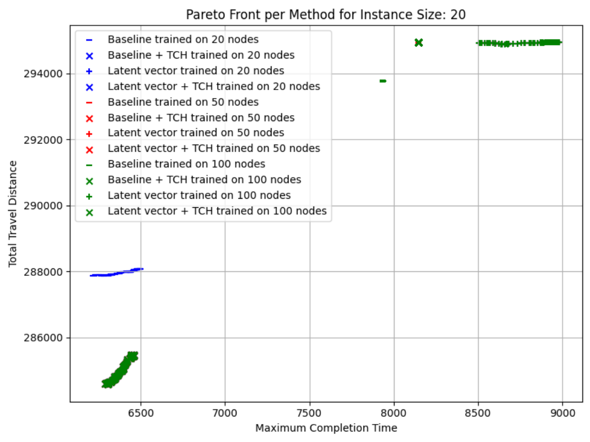


Figure 4.9: The Pareto fronts of the implemented models trained on various instance sizes evaluated on instances of 20 nodes

For instances of size 20 the baseline model trained on 20 nodes is able to generate solutions with the smallest maximum completion time, but the baseline models trained on 50 nodes and 100 nodes using TCH scalarisation are able to find solutions equivalent to the baseline model with TCH scalarisation trained on 20 nodes (bottom left on figure 4.9). The baseline models with TCH scalarisation are able to find solutions with a smaller maximum completion time. So, depending on the preference between the objectives, either the baseline model or the baseline with TCH scalarisation is more suitable. The baseline models without TCH scalarisation trained on 50 and 100 nodes are not able to generate solutions of similar objectives scores as the baseline trained on 20 nodes, their solutions are clustered on the dashed line (-) to the left of (8000, 294000) on figure 4.9. However, the latent vector models with and without TCH scalarisation trained on larger instances are able to generate solutions similar to the latent vector models trained on 20 nodes (see fronts in the top right corner of figure 4.9). Thus, based on figure 4.9, the models trained on larger instances are able to perform comparably well on smaller instances, resulting in similar Pareto fronts as the models trained on 20 nodes in most cases, except for the baseline model without TCH scalarisation. In practice, the training times themselves play an important role as well, training the larger models requires significantly more computational power and time. Often more than double (for 50 nodes) or triple (for 100 nodes) the time based on the data in table 4.1. Given the additional computational resources needed to train the models on larger instances combined with the fact that the models are not performing better than the models trained on 20 nodes, there would be no reason to train models on larger instances only to use them on smaller instances.

4.2.2. Instances of 50 nodes

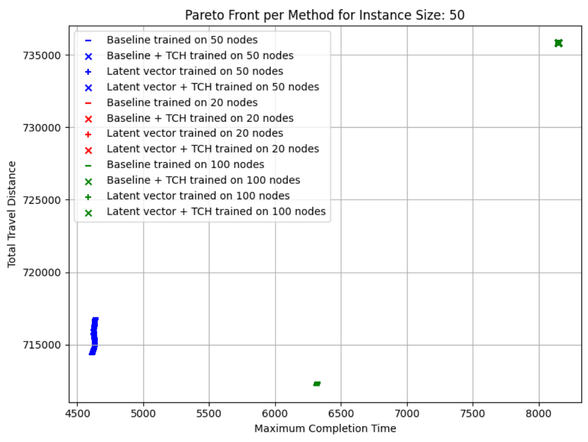


Figure 4.10: The Pareto fronts of the implemented models trained on various instance sizes evaluated on instances of 50 nodes

Similar to instances of size 20, when considering instances consisting of 50 nodes, the solutions of all of the latent vector models are very similar and clustered together. These solutions can be found in the top right of the graph in figure 4.10. The solutions found by the baseline model trained on 20 nodes are also clustered around this position. The baseline model trained on 100 nodes is able to find solutions that perform slightly better than the baseline model trained on 50 nodes for the total travel distance objective, but these solutions are significantly worse on the maximum completion time objective. So, if one were to prefer the total travel distance objective very strongly compared to the maximum completion time objective, then it could be an option to use the baseline model trained on 100 nodes rather than 50 nodes. However, this would be at the cost of 1.65 times increased training time based on table 4.1. Thus, the choice between the two baseline models depends on the available computational resources and preference between the two objectives. If computational resources are very limited, then one could consider training the models on 20 nodes as these are able to perform equally well as the other models trained on 50 nodes, at the cost of optimality relative to the baseline model trained on 50 nodes.

4.2.3. Instances of 100 nodes

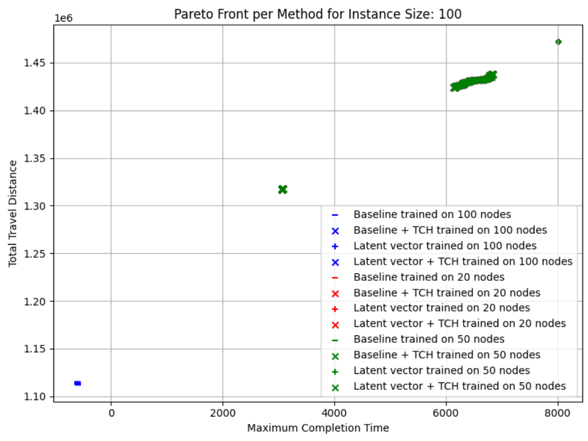


Figure 4.11: The Pareto fronts of the implemented models trained on various instance sizes evaluated on instances of 100 nodes

Also for the instances of size 100 nodes, the solutions of the latent vector models are clustered around the spots. For the latent vector models using the TCH scalarisation the solutions are grouped around a maximum completion time of 3000 (figure 4.11). While the solutions of the latent vector models not using the TCH scalarisation are grouped around a maximum completion time of 8000 (figure 4.11). The maximum completion time of the solutions found by the baseline model trained on 100 nodes are dominated by the negative time reset that is applied whenever a vehicle visits the depot node, which results in a negative objective value for the completion time, so these solutions cannot be considered when comparing against them. So, for the instances consisting of 100 nodes, the latent vector models using TCH scalarisation are able to perform the best out of the implemented models, regardless of what size instances they have been trained on. Also, the baseline models with and without TCH scalarisation trained on the smaller instances generate similar solutions to the baseline model with TCH scalarisation trained on 100 instances. Based on figure 4.11 the models trained on smaller instances are able to generalise to larger unseen instances.

4.3. Evaluation Results

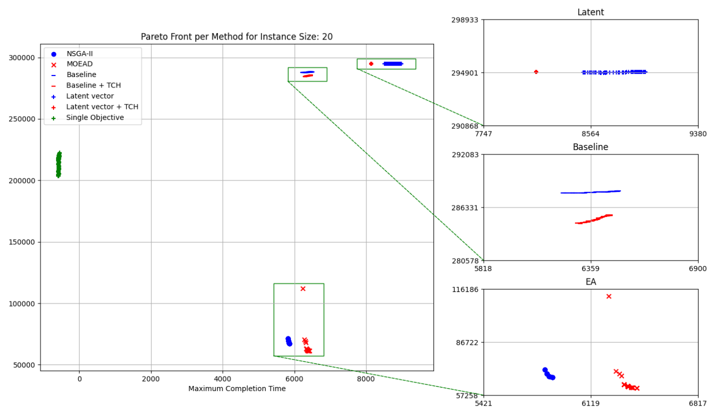
The implementation of the evolutionary algorithms (EAs), as discussed in Chapter 3, were run on instances with 20, 50, 100, and 200 nodes. This section compares the results of the EAs, NSGA-II and MOEA/D, with the implemented multi-objective deep reinforcement learning models, and the single-objective model as implemented in the paper by Wu, Zhou, Xia, *et al.* [5]. The results are shown in figure 4.12, which shows the resulting Pareto fronts of each of the models, and the two evolutionary algorithms for each of the instance sizes. The graphs in figure 4.12 have also been included in Appendix A in larger sizes.

Based on the graphs in figure 4.12, a general trend that is noticeable is that the NSGA-II algorithm is able to find solutions with relatively small maximum completion times in comparison to the MOEA/D algorithm. The MOEA/D algorithm, on the other hand, is able to find solutions with smaller total travel distances than the NSGA-II algorithm over all instance sizes that we evaluated. The spread in the Pareto fronts of both evolutionary algorithms is minimal. The fronts tend to start clustering, especially for larger instances of the problem (100 nodes and larger). The explorative aspect of these algorithms (mutation, crossover, etc.) likely suffer due to the increased search space sizes of these larger instances. It becomes harder to find feasible, Pareto optimal, solutions that differ a lot from each other. The Pareto front of the MOEA/D algorithm starts to cluster sooner, as can be seen from graph 4.12b, for 50 nodes the Pareto front already has significantly less spread than for 20 nodes (graph 4.12a). From 50 nodes and larger the spread along the Pareto front does not change significantly for the MOEA/D algorithm, indicating quite consistent performance over larger instances.

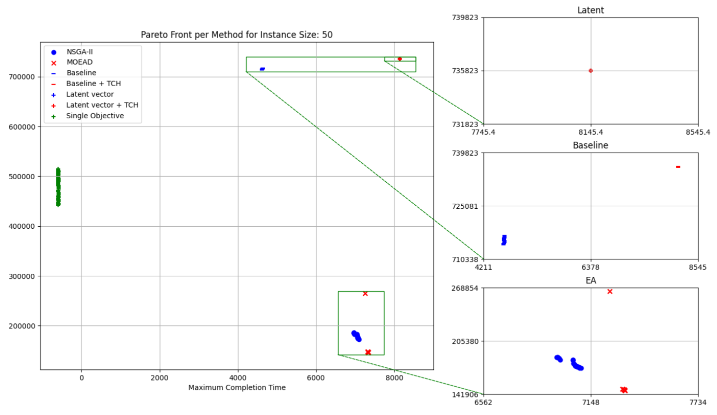
The baseline multi-objective model is able to find Pareto fronts with smaller objective scores for both objectives compared to its latent vector counterparts, except for instances of 100 nodes, where the latent vector model with TCH scalarisation performs better than the baseline with TCH scalarisation (see figure 4.12c). However, for both the baseline models and the latent vector models it holds that the spread in the Pareto front diminishes as the instance sizes start to increase. This could either be due to a lack of variance in the structure of the instances in the validation dataset, or because of the increasing complexity with larger instance sizes. Interestingly, the Pareto fronts of the baseline models have a linear structure, indicating a possible codependence between the two objectives, as an increase in total travel distance results in an increase in maximum completion time. However, this is not sufficient evidence to confirm this relationship between the two objectives, as the other fronts do not share the same structure. This is likely due to the fact that in the baseline models a simple weighted linear combination of the two objective scores is taken, using the preference vectors as the weights, which is why the eventual linear combination of the two objective scores manifests in this linear structure of the Pareto fronts. Based on the Pareto fronts generated by the models using TCH scalarisation and the variants using the weighted sum approach, there is not sufficient evidence to support or refute hypothesis 1, in which we stated that TCH scalarisation should allow us to more accurately and consistently produce Pareto fronts regardless of its shape. The generated Pareto fronts of both variants of the respective models, i.e. TCH scalarisation and weighted sum of both the baseline and the latent vector approach, are neither consistently more spread out nor do they have the same shape consistently over the varying instance sizes. Therefore, hypothesis 1 with regards to the potential improvement by applying TCH scalarisation cannot be confirmed or refuted.

For all instance sizes the single-objective model, as implemented in the paper by Wu, Zhou, Xia, *et al.* [5], is able to perform better with regards to the total travel distance objective, but the maximum completion time objective score is dominated by the model's negative reset for visiting the depot node, resulting in a negative maximum completion time. So, it is not able to generate a Pareto front that considers the trade offs between the objectives, which is to be expected. Given the added complexity of a second objective, the multi-objective models that have been implemented in this research are currently not able to outperform the single-objective model with respect to the objective score that the model was trained on.

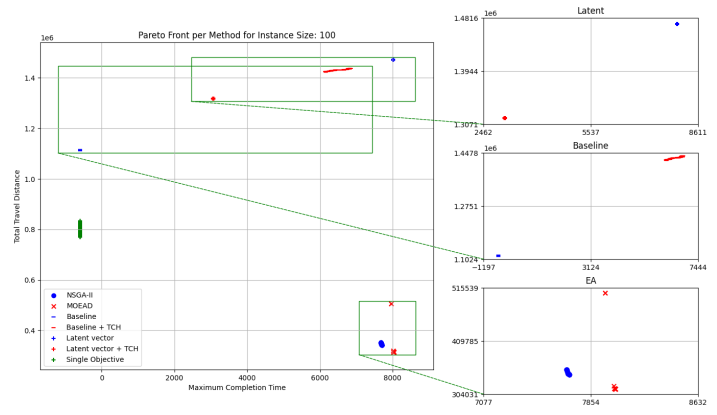
Notably, the EAs seem to perform extremely well on the total travel distance objective, but that is most likely due to the relaxation that was applied in the EA model to allow for proper Pareto fronts to be found within reasonable time. The relaxation that was applied in the EA model was to issue an arbitrary non-linear penalty for precedence constraint violations rather than disregarding the solution. As a result, the EAs were able to generate well spread out Pareto fronts at the cost of guaranteed feasibility.



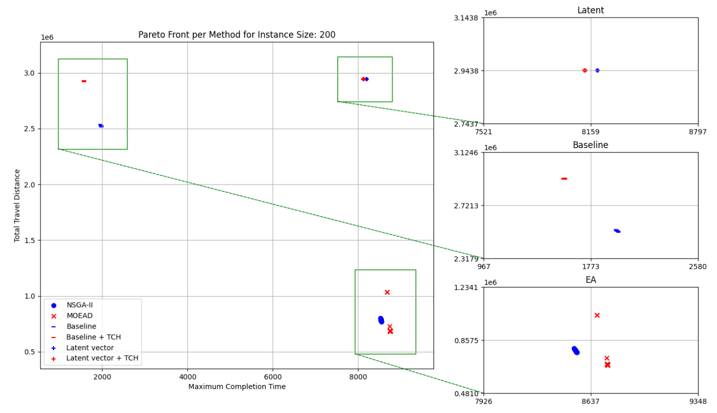
(a) The Pareto fronts for instances with 20 nodes



(b) The Pareto fronts for instances with 50 nodes



(c) The Pareto fronts for instances with 100 nodes



(d) The Pareto fronts for instances with 200 nodes

Figure 4.12: The Pareto fronts for each approach per instance size

4.4. Summary

The training performance across different instance sizes reveals that Tchebycheff (TCH) scalarisation generally improves stability and convergence of both baseline and latent vector models, particularly for smaller instances. TCH smooths the learning curves, and reduces the variance both within an epoch and between epochs. However, its benefits diminish as the instance size grows, resulting in increased variance and inconsistent convergence behaviour on larger instances. On average, over 100 epochs, the latent vector models train faster and more consistently than their baseline counterparts, with 10.48% and 4.51% lower training times for the standard and TCH variants, respectively. This allows for either quicker training to comparable performance or longer training within the same time budget. Additionally, the latent models exhibit lower variance in training times, indicating more reliable training durations. Additionally, it confirms hypothesis 2, made in Chapter 3 section 3.1.

To determine the ability of the models to generalise to unseen instances, the models trained on other instance sizes were compared against those trained on the respective instance sizes. Overall, latent vector models exhibit strong generalisation across instance sizes, with smaller models often offering favourable trade-offs between performance and computational cost, while baseline models without TCH scalarisation tend to specialise on instances they have been trained on. The solutions of these baseline models on unseen instance sizes are not similar to the same baseline models trained on the respective instance sizes.

Based on the evaluations, which compare evolutionary algorithms (NSGA-II, MOEA/D) with multi- and single-objective DRL models on instances of size 20, 50, 100, and 200 nodes, NSGA-II finds lower maximum completion times than MOEA/D, while MOEA/D excels at minimizing total travel distance. As instance sizes grow, all methods show reduced Pareto front spread, with MOEA/D clustering stronger than NSGA-II. The baseline models generally outperform the latent vector models, except at 100 nodes. The baseline's linear Pareto fronts might suggest objective coupling, but is likely due to the weighted combination of the objectives. The generated Pareto fronts of the baseline and the latent vector model variants, i.e. with TCH scalarisation and weighted sum approach, do not provide sufficient evidence to support or deny hypothesis 1, made in section 3.1. The single-objective model excels in optimising the total travel distance, but fails on maximum completion time due to its reward structure. The strong performance of the EAs, especially on total travel distance, is attributed to relaxations in constraint handling, allowing better Pareto front coverage at the expense of feasibility guarantees.

5 Conclusion & Future Work

5.1. Conclusion

By combining the multi-objective and preference conditioned aspect of the model described by Lin, Yang, and Zhang [9], with the deep reinforcement learning (DRL) model created to learn single-objective optimisation of AGH instances, as implemented in Wu, Zhou, Xia, *et al.* [5], the deep reinforcement learning model has been extended to solve for multiple objectives. The objectives that were optimised are the total travel distance of all the vehicles, and the maximum completion time of the latest task. The initial combination of the models, which is named the baseline, applied a linear weighted combination of the multiple objectives to create a scalar reward that was used during learning. Potential improvements that were added to this initial combination were the use of Tchebycheff (TCH) scalarisation of the two objectives, to ensure all parts of any type of Pareto fronts can be learned. Additionally, latent vectors were used to reduce computational complexity, possibly at the cost of output accuracy.

The retrieved training curves (section 4.1) show that TCH scalarisation improves the stability and convergence of both the baseline and latent vector models, especially for smaller instances. Furthermore, the collected data in table 4.1 proofs that the latent vector models are able to train in significantly less time than the baseline models. Overall, the latent vector models do not generate the most optimal Pareto fronts, but they do tend to generalise better to unseen instances than the baseline models.

Since truth labels or optimal values were unavailable for the existing instances, relative accuracy is used to compare various methods. The implemented multi-objective DRL were evaluated against Evolutionary Algorithms NSGA-II and MOEA/D, and the existing single-objective DRL model. While the single-objective DRL model performed best with regards to total travel distance, it lacks the ability to find trade off points between the two objectives. NSGA-II finds lower maximum completion times of the two EAs, while MOEA/D finds solutions with smaller total travel distances. As instance sizes increase, the Pareto fronts generated by all methods have reduced spread compared to smaller instances, likely due to greater complexity and larger search space. The implemented multi-objective models are able to find trade off points between the objectives, with slightly worse solutions than the single-objective model with respect to the total travel distance objective, while still guaranteeing feasibility, unlike the EA algorithms which have been relaxed to ensure proper Pareto front generation within reasonable time.

Although the Pareto fronts of some of the baseline models have a linear shape, which might suggest interdependence between the objectives, it is most likely due to linear nature of the logic used in combining the values in the baseline models. Predominantly, the Pareto fronts generated by the models do not show a strong relationship between the two objectives, meaning the objectives are independent of each other.

5.2. Future Work

Currently time within the model is determined by calculations using the travel distances, vehicle speed, and waiting time. This time is reset whenever a vehicle arrives at the depot node by adding some negative value to its time. This negative time reset dominates the values of the completion time objective during training of almost all the models, especially on larger instance sizes where vehicles visit the depot node more often. In some cases, for example the baseline model on 100 nodes, the solutions generated by the models is also affected by this. However, generally the models were able to find feasible solutions to the evaluation instances. To prevent this from happening the computation of the completion time objective should be updated to use a moving maximum and track the maximum time encountered before the resets are applied. An entirely different way of tracking the time could also be introduced, for example a method in which the time is tracked independent of the vehicles and travel distances, but rather in a global sense.

Furthermore, the EAs that were implemented during this thesis were relaxed by removing the strict precedence constraint and introducing an arbitrary non-linear penalty based on the number of constraint violations. So, in future work, where there are less strict time limits, the precedence constraint should be applied. Also, the focus of this research was not the implementation and optimisation of EAs for multi-objective AGH, so a framework was used that allows you to apply out-the-box EAs to your problem definition. These out-the-box algorithms do provide the option of tweaking parameters such as mutation, crossover, number of generations, and so on. However, the algorithms will of course not be optimised for AGH specifically as this requires (extensive) domain knowledge, about both AGH problems and EAs, to implement custom operators for the respective evolutionary algorithms. Thus, in the future, the implemented EAs could be optimised further to provide a more accurate comparison with the neural models.

A
Separate evaluation result graphs

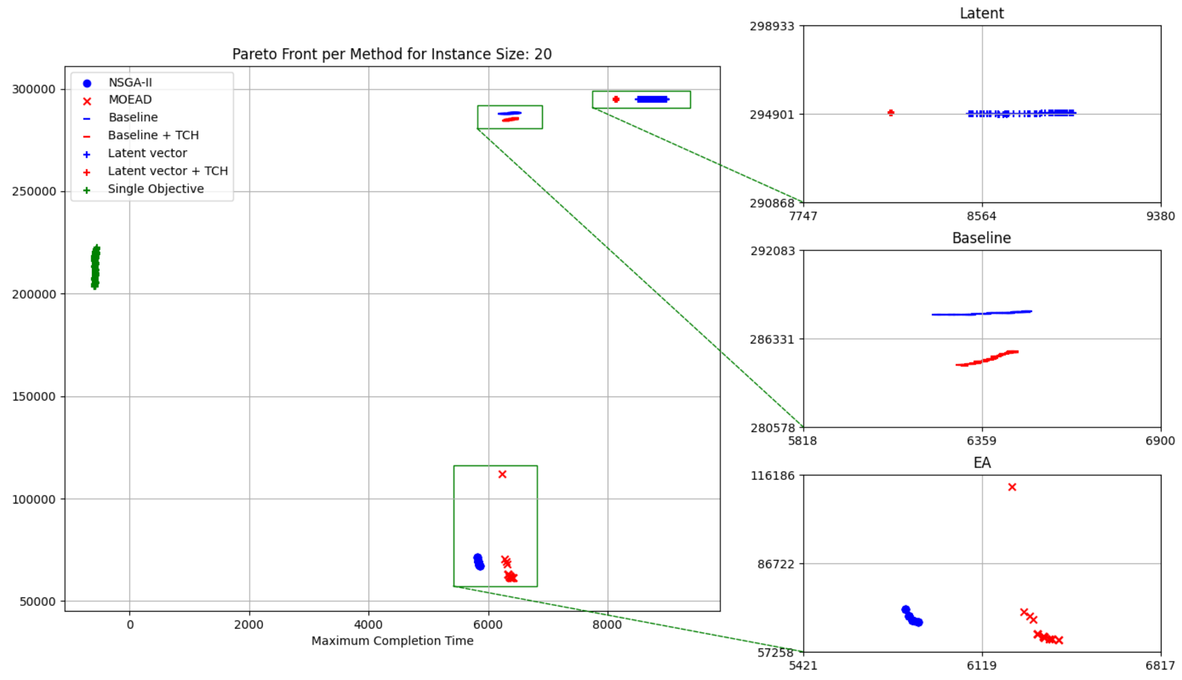


Figure A.1: The Pareto fronts for instances with 20 nodes

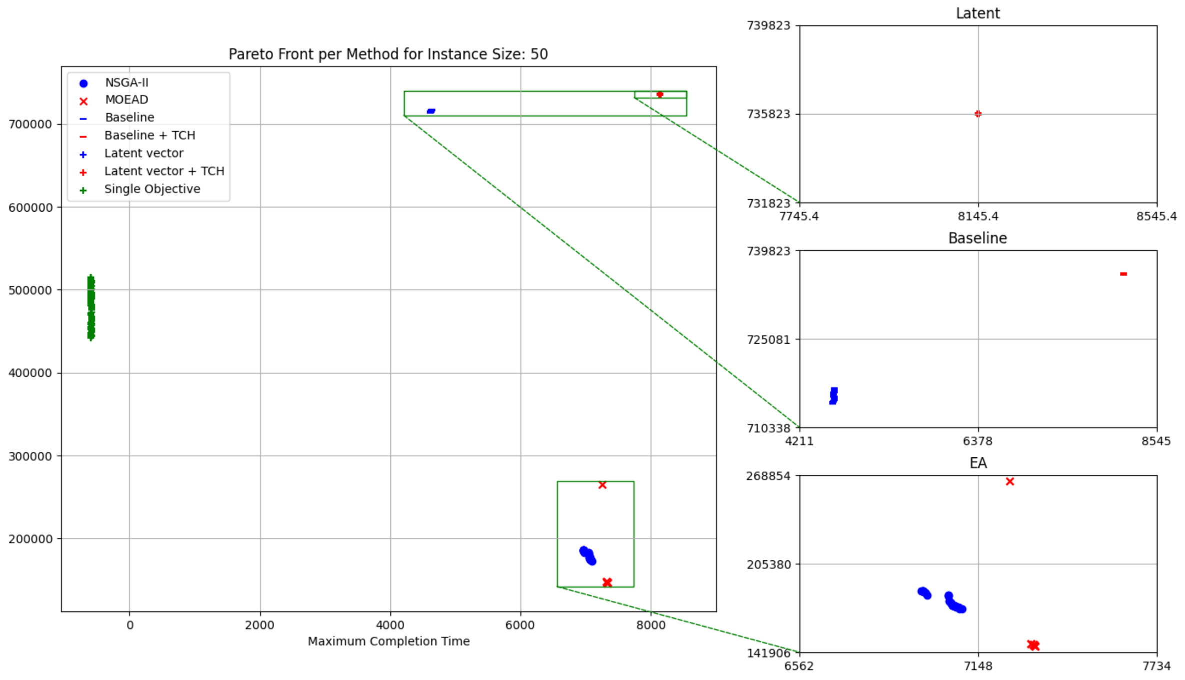


Figure A.2: The Pareto fronts for instances with 50 nodes

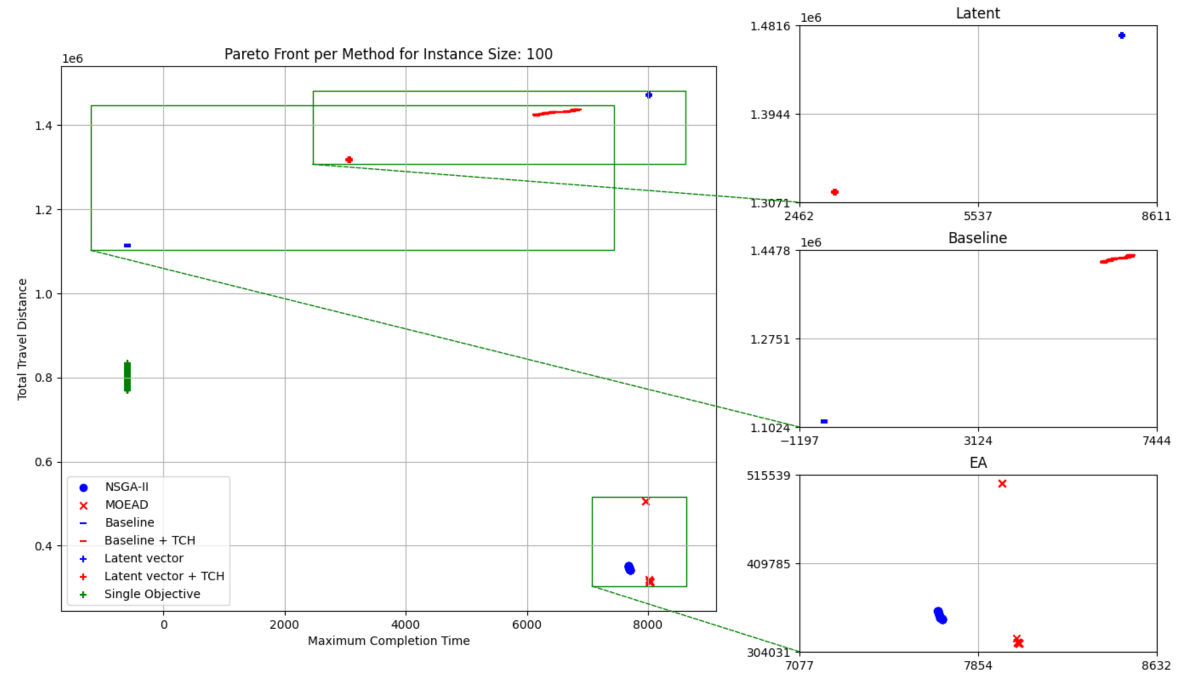


Figure A.3: The Pareto fronts for instances with 100 nodes

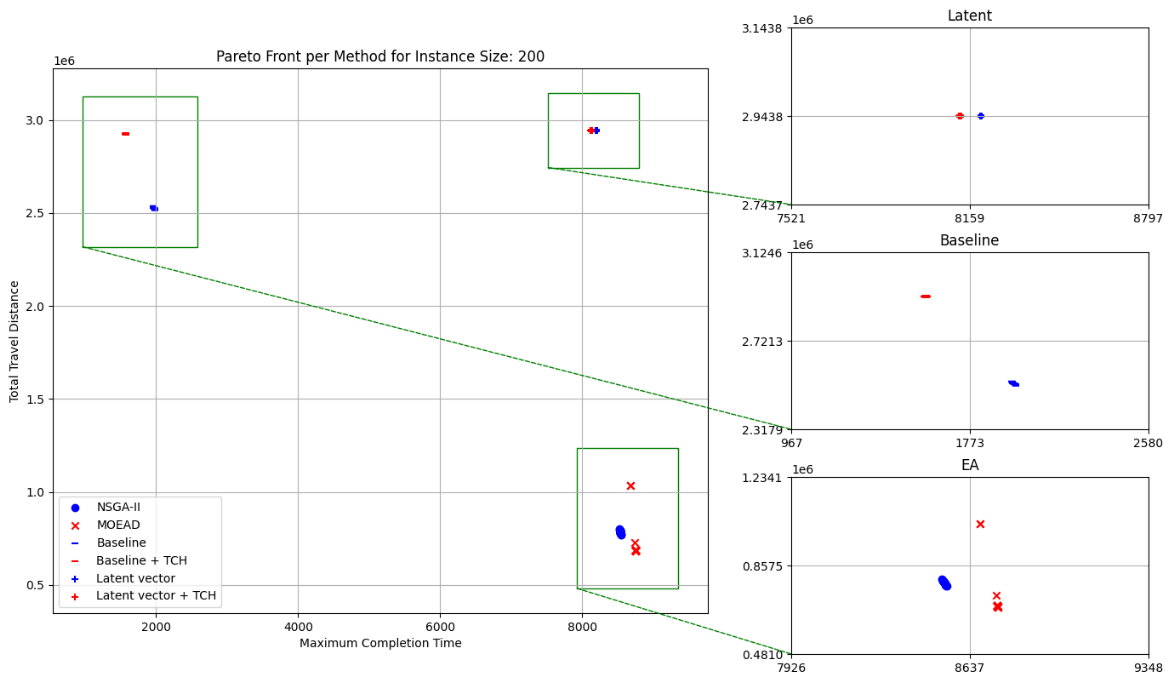


Figure A.4: The Pareto fronts for instances with 200 nodes

References

[1] I. A. T. A. (IATA), *Air cargo market analysis october 2024*, [Accessed 04-12-2024], 2024. [Online]. Available: `https://www.iata.org/en/iata-repository/publications/economic-reports/air-cargo-market-analysis-october-2024/`.

[2] I. A. T. A. (IATA), *Air passenger market analysis october 2024*, [Accessed 04-12-2024], 2024. [Online]. Available: `https://www.iata.org/en/iata-repository/publications/economic-reports/air-passenger-market-analysis-october-2024/`.

[3] S. Padrón, D. Guimarans, J. J. Ramos, and S. Fitouri-Trabelsi, "A bi-objective approach for scheduling ground-handling vehicles in airports," *Computers & Operations Research*, vol. 71, pp. 34–53, 2016, ISSN: 0305-0548. DOI: `https://doi.org/10.1016/j.cor.2015.12.010`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0305054815002968`.

[4] *EUROCONTROL CODA Digest - All-Causes Delays to Air Transport in Europe - Annual 2022*. eurocontrol.int, 2022.

[5] Y. Wu, J. Zhou, Y. Xia, X. Zhang, Z. Cao, and J. Zhang, "Neural airport ground handling," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 12, pp. 15 652–15 666, 2023. DOI: `10.1109/TITS.2023.3253552`.

[6] M. Ehrgott and X. Gandibleux, "An annotated bibliography of multiobjective combinatorial optimization," *Operations Research-Spektrum*, vol. 22, pp. 425–460, Nov. 2000. DOI: `10.1007/s002910000046`.

[7] M. Ehrgott, *Multicriteria optimization*, 2005.

[8] A. Herzel, S. Ruzika, and C. Thielen, "Approximation methods for multiobjective optimization problems: A survey," *INFORMS Journal on Computing*, Feb. 2021. DOI: `10.1287/ijoc.2020.1028`.

[9] X. Lin, Z. Yang, and Q. Zhang, *Pareto set learning for neural multi-objective combinatorial optimization*, 2022. arXiv: `2203.15386 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2203.15386`.

[10] F. Emmert-Streib and M. Dehmer, "Taxonomy of machine learning paradigms: A data-centric perspective," *WIREs Data Mining and Knowledge Discovery*, vol. 12, no. 5, e1470, 2022. DOI: `https://doi.org/10.1002/widm.1470`. eprint: `https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1470`. [Online]. Available: `https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1470`.

[11] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4.

[12] P. Flach, *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012.

[13] J. Friedman, "The elements of statistical learning: Data mining, inference, and prediction," *(No Title)*, 2009.

[14] L. P. Kaelbling, M. L. Littman, and A. W. Moore, *Reinforcement learning: A survey*, 1996. arXiv: `cs/9605103 [cs.AI]`. [Online]. Available: `https://arxiv.org/abs/cs/9605103`.

[15] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete event dynamic systems*, vol. 13, no. 4, pp. 341–379, 2003.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

[17] D. Silver, *Lectures on reinforcement learning*, url: `https://www.davidsilver.uk/teaching/`, 2015.

[18] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” en, *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[19] Y. Bengio, A. Courville, and P. Vincent, *Representation learning: A review and new perspectives*, 2014. arXiv: 1206.5538 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1206.5538.

[20] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015, issn: 0893-6080. doi: 10.1016/j.neunet.2014.09.003. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2014.09.003.

[21] T. Alashkar, *Real-time lane and car detection on highway using vision system*, Dec. 2016. doi: 10.13140/RG.2.2.18416.05128.

[22] M. Zhu, J. Gu, B. Chen, and P. Gu, “Dynamic subcarrier assignment in ofdma-pons based on deep reinforcement learning,” *IEEE Photonics Journal*, vol. 14, pp. 1–11, Apr. 2022. doi: 10.1109/JPHOT.2022.3148259.

[23] L. T. Mohammad Harun Rashid, “Parallel Combinatorial Optimization Heuristics with GPUs,” *Advances in Science, Technology and Engineering Systems Journal*, vol. 3, no. 6, pp. 265–280, 2018. doi: 10.25046/aj030635.

[24] *Solving combinatorial optimization and np-hard problems using novel quantum annealing using nec vector engine*, https://hprc.tamu.edu/files/aces_NEC_quantum.pdf.

[25] A. Gupta and S. Saini, “An enhanced ant colony optimization algorithm for vehicle routing problem with time windows,” in *2017 Ninth International Conference on Advanced Computing (ICoAC)*, Dec. 2017, pp. 267–274. doi: 10.1109/ICoAC.2017.8441175.

[26] G. B. Dantzig and J. H. Ramser, “The truck dispatching problem,” *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[27] D. Tas, “Time and reliability in vehicle routing problems,” 2013.

[28] C. G. Corlu, R. de la Torre, A. Serrano-Hernandez, A. A. Juan, and J. Faulin, “Optimizing energy consumption in transportation: Literature review, insights, and research opportunities,” *Energies*, vol. 13, no. 5, 2020, issn: 1996-1073. doi: 10.3390/en13051115. [Online]. Available: https://www.mdpi.com/1996-1073/13/5/1115.

[29] K. Sindhya, “An introduction to multiobjective optimization,” *SAS julkaisusarja*, 2016.

[30] K. Mason and E. Howley, “Evolving multi-objective neural networks using differential evolution for dynamic economic emission dispatch,” in *the Genetic and Evolutionary Computation Conference Companion*, Jul. 2017, pp. 1287–1294. doi: 10.1145/3067695.3082480.

[31] K. Florios and G. Mavrotas, “Generation of the exact pareto set in multi-objective traveling salesman and set covering problems,” *Applied Mathematics and Computation*, vol. 237, pp. 1–19, 2014, issn: 0096-3003. doi: https://doi.org/10.1016/j.amc.2014.03.110. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0096300314004810.

[32] C. Papadimitriou and M. Yannakakis, “On the approximability of trade-offs and optimal access of web sources,” in *IEEE Symposium on Foundations of Computer Science (FOCS*, Feb. 2000, pp. 86–92, isbn: 0-7695-0850-2. doi: 10.1109/SFCS.2000.892068.

[33] A. Jaszkiewicz, “Genetic local search for multi-objective combinatorial optimization,” *European Journal of Operational Research*, vol. 137, no. 1, pp. 50–71, 2002, issn: 0377-2217. doi: https://doi.org/10.1016/S0377-2217(01)00104-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221701001047.

[34] Q. Zhang and H. Li, “Moea/d: A multiobjective evolutionary algorithm based on decomposition,” *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007. doi: 10.1109/TEVC.2007.892759.

[35] M. Ehrgott and X. Gandibleux, “Hybrid metaheuristics for multi-objective combinatorial optimization,” in Springer, Jun. 2008, vol. 114, pp. 221–259, isbn: 978-3-540-78294-0. doi: 10.1007/978-3-540-78295-7_8.

[36] Y. Bengio, A. Lodi, and A. Prouvost, “Machine learning for combinatorial optimization: A methodological tour d’horizon,” *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021, issn: 0377-2217. doi: https://doi.org/10.1016/j.ejor.2020.07.063. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221720306895.

[37] M. Kruber, M. Lübbecke, and A. Parmentier, “Learning when to use a decomposition,” in *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, May 2017, pp. 202–210, isbn: 978-3-319-59775-1. doi: 10.1007/978-3-319-59776-8_16.

[38] P. Bonami, A. Lodi, and G. Zarpellon, “Learning a classification of mixed-integer quadratic programming problems,” in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018*, Berlin, Heidelberg: Springer-Verlag, 2018, isbn: 978-3-319-93030-5. doi: 10.1007/978-3-319-93031-2_43. [Online]. Available: https://doi.org/10.1007/978-3-319-93031-2_43.

[39] A. Lodi and G. Zarpellon, “On learning and branching: A survey,” *TOP*, vol. 25, pp. 1–30, Jun. 2017. doi: 10.1007/s11750-017-0451-6.

[40] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, *Exact combinatorial optimization with graph convolutional neural networks*, 2019. arXiv: 1906.01629 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1906.01629.

[41] X. Chen and Y. Tian, *Learning to perform local rewriting for combinatorial optimization*, 2019. arXiv: 1810.00337 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1810.00337.

[42] A. Nowak, S. Villar, A. S. Bandeira, and J. Bruna, *Revised note on learning algorithms for quadratic assignment with graph neural networks*, 2018. arXiv: 1706.07450 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1706.07450.

[43] P. Emami and S. Ranka, *Learning permutations with sinkhorn policy gradient*, 2018. arXiv: 1805.07010 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1805.07010.

[44] E. Larsen, S. Lachapelle, Y. Bengio, E. Frejinger, S. Lacoste-Julien, and A. Lodi, “Predicting tactical solutions to operational planning problems under imperfect information,” *INFORMS Journal on Computing*, vol. 34, no. 1, pp. 227–242, Jan. 2022, issn: 1526-5528. doi: 10.1287/ijoc.2021.1091. [Online]. Available: http://dx.doi.org/10.1287/ijoc.2021.1091.

[45] O. Vinyals, M. Fortunato, and N. Jaitly, “Pointer networks,” in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf.

[46] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, *Neural combinatorial optimization with reinforcement learning*, 2017. arXiv: 1611.09940 [cs.AI]. [Online]. Available: https://arxiv.org/abs/1611.09940.

[47] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1706.03762.

[48] W. Kool, H. van Hoof, and M. Welling, *Attention, learn to solve routing problems!* 2019. arXiv: 1803.08475 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1803.08475.

[49] Y. Liu, J. Wu, J. Tang, W. Wang, and X. Wang, “Scheduling optimisation of multi-type special vehicles in an airport,” *Transportmetrica B: Transport Dynamics*, vol. 10, no. 1, pp. 954–970, Oct. 2021, issn: 2168-0582. doi: 10.1080/21680566.2021.1983484. [Online]. Available: http://dx.doi.org/10.1080/21680566.2021.1983484.

[50] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine Learning*, vol. 8, no. 3, pp. 229–256, May 1992, issn: 1573-0565. doi: 10.1007/BF00992696. [Online]. Available: https://doi.org/10.1007/BF00992696.

[51] K. Li, T. Zhang, and R. Wang, “Deep reinforcement learning for multiobjective optimization,” *IEEE Transactions on Cybernetics*, vol. 51, no. 6, pp. 3103–3114, Jun. 2021, issn: 2168-2275. doi: 10.1109/tcyb.2020.2977661. [Online]. Available: http://dx.doi.org/10.1109/TCYB.2020.2977661.

[52] H. Wu, J. Wang, and Z. Zhang, *Modrl/d-am: Multiobjective deep reinforcement learning algorithm using decomposition and attention model for multiobjective optimization*, 2020. arXiv: 2002.05484 [cs.NE]. [Online]. Available: https://arxiv.org/abs/2002.05484.

[53] Y. Zhang, J. Wang, Z. Zhang, and Y. Zhou, *Modrl/d-el: Multiobjective deep reinforcement learning with evolutionary learning for multiobjective optimization*, 2021. arXiv: 2107.07961 [cs.NE]. [Online]. Available: https://arxiv.org/abs/2107.07961.

[54] Z. Zhang, Z. Wu, H. Zhang, and J. Wang, “Meta-learning-based deep reinforcement learning for multiobjective optimization problems,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 10, pp. 7978–7991, 2022.

[55] K. Miettinen, *Nonlinear Multiobjective Optimization* (International series in operations research & management science). Springer, 2012.

[56] R. Steuer, *Multiple Criteria Optimization: Theory, Computation, and Application* ((WILEY SERIES IN PROBABILITY AND MATHEMATICAL STATISTICS)). Wiley, 1986, isbn: 9780471859703.

[57] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, “Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit,” *Nature*, vol. 405, no. 6789, pp. 947–951, Jun. 2000, issn: 1476-4687. doi: 10.1038/35016072. [Online]. Available: https://doi.org/10.1038/35016072.

[58] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002. doi: 10.1109/4235.996017.

[59] A. Nebro, J. Galeano-Brajones, F. Luna, and C. Coello, “Is nsga-ii ready for large-scale multiobjective optimization?” *Mathematical and Computational Applications*, vol. 27, p. 103, Nov. 2022. doi: 10.3390/mca27060103.

[60] J. Blank and K. Deb, “Pymoo: Multi-objective optimization in python,” *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.

[61] D. H. P. C. C. (DHPC), *DelftBlue Supercomputer (Phase 2)*, https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2, 2025.