Bachelor Thesis

Non-Stationarity in the Train-Scheduling-Problem: Leveraging Effects of Curriculum Learning

Étienne Künzel

Matrikelnummer: *7477641*

Supervisor: *Prof.Dr.Visvanathan Ramesh*

Goethe University of Frankfurt

March 11, 2024

Abstract

Trains are a long-existing medium of transportation and supply chain management, which are still of the utmost importance for global and local transportation of goods and individuals to the present day. With the significance of trains, this bachelor thesis delves into the intricacies of train scheduling, the problem of controlling multiple trains, and adapting to unforeseen problems.
While Operations Research methodologies currently dominate train scheduling, their limitations in adaptability and computational efficiency prompt exploration into alternative approaches.
This thesis investigates the application of Multi-Agent Reinforcement Learning in the train scheduling problem and highlights the advantages of designing training curricula derived from a deconstruction of the train scheduling problem.
By utilizing this Custom Curriculum, we were able to improve the mean done rate(number of trains reaching their destination) of a DDDQN algorithm by about 160% compared to using No Curriculum.
We further explore adaptations to the DDDQN addressing the Non-Stationarity, which is introduced with the changing environments of the curricula to leverage the positive effects of the custom curricula.
The adaptation that improved mean done rate the most, when evaluated in an environment not being part of the training data, was the utilization of rational padé activation units(a type of learnable activation function), which increased the mean done rate by roughly 232%, but also the use of elastic weight consolidation yielded an improvement of 195%, both showing us that we are able to leverage the effects of a curriculum by using adaptations to Non-Stationarity, commonly used in continual/lifelong reinforcement learning setting.
The insights gained contribute to making RL more applicable to logistics and supply chain management tasks, enhancing efficiency and adaptability across them, but they need further investigation due to the results displaying high variance. diverse scenarios.

Table of contents

Contents

1 Introduction

The modern world relies heavily on a complex network of different transportation systems to enable fast and reliable movement of goods and people alike. In this context, the efficient scheduling and organization of multiple trains play a pivotal role in ensuring the smooth functioning of the transportation infrastructure. In comparison to other means of transportation, rail traffic is very environmentally friendly. Additionally, the European Union funded different railway projects with more than half a billion Euros to ensure a functioning and future-proof railway system in the EU[1].
The Train Scheduling Problem (TSP) emerges as a challenge in this domain, comprising the optimization of multiple train paths, resources, and timetables while adapting to unforeseen events by dynamically replanning routes. You can view it as merging the Multi-Agent-Pathfinding (MAPF) and Vehicle Rescheduling Problem (VRSP).
Currently, train scheduling relies heavily on established Operations Research (OR) methodologies. Despite offering several advantages, these methods are not without their challenges.
Due to the characteristics of the TSP, we can deconstruct the issue into a graph optimization problem, in which we need to find non-overlapping routes in time and location through the network. OR is proficient at addressing the complexities of the train scheduling problem by utilizing mathematical modeling and optimization techniques. Its ability to systematically optimize the paths of the trains allows OR to formulate effective and efficient schedules that enhance overall railway system performance[2].
But these advantages do not come without tradeoffs. One significant drawback lies in the limited flexibility and speed to react to unforeseen factors such as incidents, weather, and broken tracks that require rescheduling one or more trains.
Moreover, with the expansion of railway networks and the growing number of agents, the computational requirements of OR methods are increasing exponentially, leading to a high computational expenditure to calculate optimal schedules for larger systems, particularly when numerous reschedulings are necessary.
The surge in interest surrounding the utilization of Machine Learning (ML), and especially Reinforcement Learning (RL) as an alternative approach to OR for tackling challenges within Logistics and Supply Chain Management (SCM), has led to increased research efforts, including in the domain of train transportation. With RL excelling at similar tasks like maritime and air logistics[3], it presents a promising alternative to address the TSP.
RL stands out for its efficiency in deriving actions with reduced time requirements(after training) and enhanced adaptability to uncertainties compared to OR. This adaptability positions RL as a flexible and compelling option for scenarios characterized by changing conditions and unpredictable variables, as they often appear in real-life applications.
In RL, an agent learns to make decisions by interacting with an environment. The agent receives feedback in the form of rewards and punishments based on its actions. Through trial and error, the agent learns a strategy, adjusts its actions, trying to achieve optimal decision-making and receiving the most rewards possible. This way of learning is similar to the human way of learning, where we learn by adapting our strategies based on the outcomes of previous decisions.
The RL framework builds upon a Markov Decision Process (MDP). A MDP is a mathematical construct describing the connection between states, actions, and rewards, where the possible actions and rewards depend only on the current state and nothing else, meaning that the underlying dynamics of a problem are not shifting over time.

But in the real world, many different factors introduce such time-dependent changes. This change in underlying dynamics can occur in different ways and is called Non-Stationarity. In research, Non-Stationary combined with RL is the main subject of continual or lifelong RL[4]. Learning to deal with Non-Stationarity would be an important step in deploying RL models in the real world, where dangerous situations emerge from the unexpected and can lead to severe consequences if not dealt with correctly.
To perform reliably in such environments requires the capability to acquire new abilities, build upon already learned knowledge, and disregard useless information. In stationary environments, RL models are often able to overfit to bigger environments[5], just seeming to "understand" the task and its underlying dynamics rather than learning the required abilities to solve similar tasks in other circumstances.
Besides that, we work with multiple trains/agents, meaning we are acting in a Multi-Agent-RL (MARL) setting, which not only complicates the base problem but also introduces a certain amount of Non-Stationarity through the interaction between two or more agents[6]. Each agent changes its behavior after interacting with the environment and other agents. As a result, an agent can't assume the same behavior from another agent every time they meet but instead needs to consider that the other agent will probably not act the same as before.
To enable our RL models to generalize more and therefore be able to apply skills in different, Non-Stationary environments, we can use Curriculum Learning (CL). CL has shown to be a powerful technique in RL and ML overall to help our models converge faster and generalize better[7]. In CL, we sequence and create training data in a sensible order to promote our model to learn the essence of a skill instead of letting it overfit on an environment.
However, with the introduction of a curriculum and the consequent change in environments, we also introduce an often overlooked Non-Stationarity. Considering this Non-Stationarity could leverage the positive effects CL has on RL. We can roughly categorize Non-Stationarity approaches into three groups that allow for combination[4].
Firstly, we can implement knowledge retention, where we use previous environments, model configurations, or parameters to keep past tasks in consideration.
Second, we can utilize multiple structures in one model to compensate for the changing dynamics of the problem. A common theme is the implementation of multiple approximators which alternate depending on the environment they are in. Third, we can address the Non-Stationarity by letting our models learn to learn. This is possible by modeling the Non-Stationarity with an internal representation of the RL model and utilizing this representation as the foundation for decision-making based on the current state and internal context. Another way is to use adaptive exploration, meaning letting our model explore the environment in situations after the environment changes to get a deeper understanding of the current environment.
To abstract the TSP from the real world, We use the Flatland Simulator[8] from the DB, SNCF, and SBB-CFF-FFS in collaboration with AICrowd. It simplifies the complexities of railway networks in a two-dimensional grid world where trains must reach their destinations according to various schedules and was created to bring the Flatland Challenge to life and promote the development of more efficient solutions for the TSP. Everybody was allowed to commit his solutions to win different prizes. In this competition, RL couldn't keep up with the performance of OR.
We want to tackle the following questions in this work:

1. How can we design effective curricula in the TSP?
2. How do known RL algorithms perform in the TSP?
3. How do these algorithms interact with different curricula?
4. How can we adapt our algorithms to address Non-Stationarity?
5. Are we able to leverage the positive effects of a curriculum with these adaptations?

Overall, this bachelor thesis aims to deepen the understanding of employing RL as a solution to the TSP, with a specific focus on integrating CL and handling Non-Stationarity. The goal is to analyze the effects of CL and how addressing the Non-Stationarity it introduces changes the performance and the ability to learn of different RL models. By doing so, this thesis seeks to contribute to the ongoing efforts to make RL more applicable not just to the TSP but to a broader spectrum of tasks in Logistics and SCM to make processes more efficient and adaptable in the future.
Striving for this target involves breaking down the TSP into smaller sub-problems and creating a custom-made curriculum consisting of different environments, each teaching the agent the ability to deal with one of these problems. Moreover, we will observe the influence this curriculum has on the Advantage Actor-Critic Algorithm, Proximal Policy Optimization, and the Deep Dueling Double Q-Network in comparison with a simpler curriculum and No Curriculum at all. Further, these algorithms are then adapted to address Non-Stationarity to examine the effects of this adaptation and if it is leveraging the effects of CL.
Additionally, we will adapt the given flatland reward function from a sparse reward setting to our agents receiving a reward at every timestep to improve the acquisition of skill in a shorter time period.
Gaining insight into effectively using CL, managing Non-Stationarity, and coordinating multiple actors within a network in the train scheduling context can also yield benefits for related fields, such as automated car traffic and others where similar dynamics apply.

Outline

- In Chapter 2, we give the theoretical background on RL and present three baseline algorithms.
- In Chapter 3, we talk about similar work connected to Non-Stationarity, CL, the TSP, OR, and the Flatland Challenge.
- In Chapter 4, we introduce the Flatland-Simulator, a simulator to train and test RL and OR algorithms on TSP.
- In Chapter 5, we dissect the TSP into smaller subproblems, which define all the different situations where our agents need to act the right way to not end up in suboptimal positions, from which we derive two curricula. Additionally, we introduce two simpler curricula for comparison. After that, we present three different adaptations we can make to our algorithms to adapt them to Non-Stationarity.
- In Chapter 6, the effects of the different curricula on the algorithms are examined, and how the adaptions to Non-Stationarity are changing the results and their interaction with the curricula.
- In Chapter 7, we talk about the direction the research can go in the future, the limitations and the conclusion of this work.

2 Reinforcement Learning

2.1 Definition of Reinforcement Learning

RL is one of many part in ML and Artificial Intelligence. In RL, we have an agent, also referred to as a learner in an environment. The learner can observe and take different actions in this environment. The goal of our agent is to get a reward as high as possible. The agent receives this reward depending on the action it takes in a certain state. The learner follows a so-called policy to choose a action. In Figure 1, you can see the standard improving loop of RL, where our agent interacts with its environment and receives the state and reward.

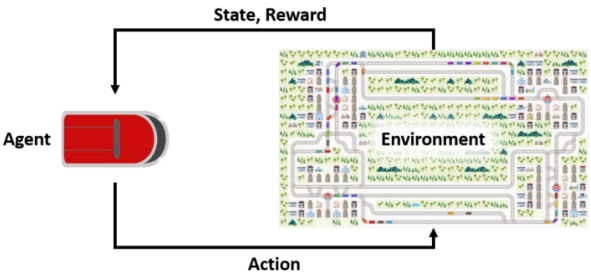


Figure 1: The basic RL framework[9]

In the start, the RL-Model tries random actions and tweaks its parameters to achieve better results (a higher reward), dependent on the inputs of the environment it receives. Through this interaction with the environment, the agent learns how to react to different situations and perform better in its surroundings.

RL is quite similar to our human learning process, so it has many use cases where humans are working on tasks or playing games that an RL model could do better. For example, in 1995, a backgammon model trained with deep-reinforcement learning was able to beat human backgammon professionals[10]. Others use cases ranging from playing chess, poker, or Starcraft 2[11] to more serious problems like detecting different illnesses in an X-ray picture[12]. Figure 2 is an overview of the possible applications for reinforcement learning.

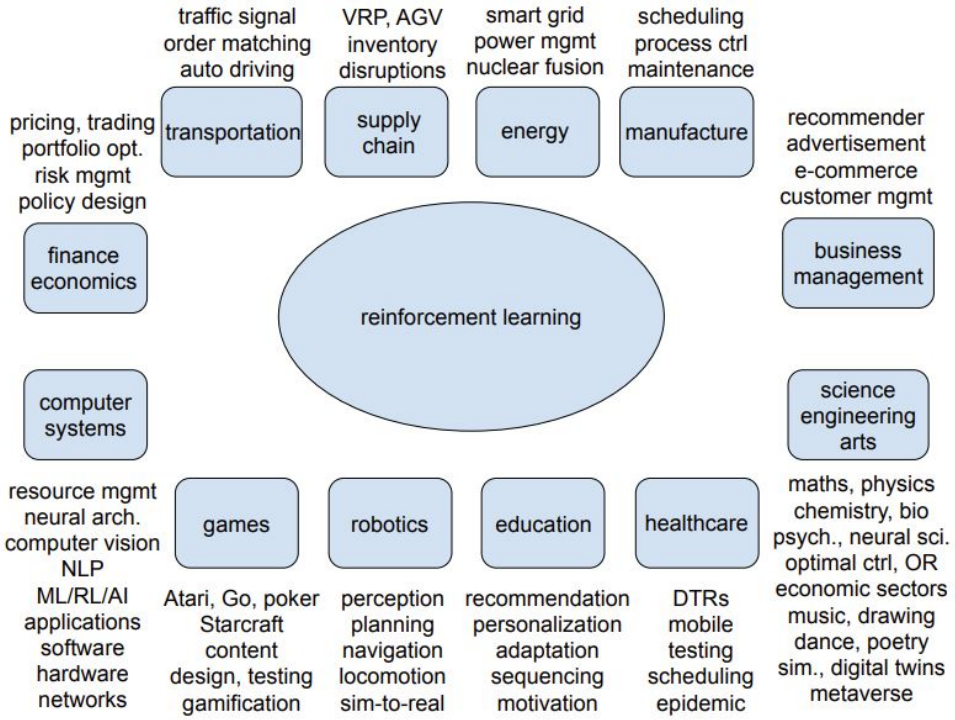


Figure 2: Reinforcement Learning and its use cases[13]

2.2 Markov-Decision-Process

A Markov Decision Process , displayed in Figure 3, involves two key components: an agent and an environment. The agent can interact with the environment by selecting an action from a set of possible actions. This action has consequences, leading to a reward and a new state provided to the agent as feedback. In most cases, the MDP is used as a basis for the reinforcement learning framework.
To illustrate this, consider a train in a railway system. If the train moves forward, its position on the grid will change, altering the state of the problem and providing a reward. To adapt the MDP for reinforcement learning it can be expressed as the following equation[3]:

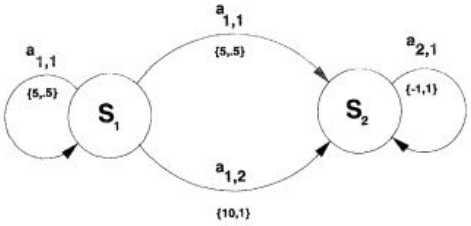


Figure 3: Symbolic representation of two state Markov process[14]

$$\max_{\pi} \lim_{T\to\infty} \mathbb{E}_{\pi} = \left[\sum_{t=0}^{T} \gamma^t r_t | s_0 = s, \pi\right] \tag{1}$$

where:

- π : A policy which concludes an action from the given states s
- t: time step
- T: infinite time horizon
- s: states s_0 starting state
- r_t: reward at timstep t
- γ^t: discount factor

The goal of equation 1is to find a policy π which leads to the highest reward possible over all timesteps T, by choosing the best possible actions a, dependent on the state s. To guarantee convergence, the discount factor γ^t reduces the weight of the reward depending on how much it lies in the future.
Continuing, the action a_t^*returning the most reward in state s with the policy π is computed over the timesteps t by the optimality condition, with $V_\pi(\cdot)$ representing the value(maximum possible reward from this state onward) of the state.[3]:

$$a_t^* = arg\, \max_{a_t} \mathbb{E}_{\pi}[r_t + \gamma V_\pi(s_{t+1})] \tag{2}$$

Furthermore, to receive the solution for a_t^*, we need to calculate the value function $V_\pi(s)$, which can be rewritten recursively using the Bellman equation, leaving us with the following equation for $V_\pi(s)$:

$$V_\pi(s_t) = \max_{a_t} \mathbb{E}_{\pi}[r_t + \gamma V_\pi(s_{t+1})] \tag{3}$$

To obtain a_t^*, we can insert Equation (3) into Equation (2). To calculate the true value $V_\pi(s)$ in Equation (3), we need to consider all reachable states combined with all possible actions and calculate them backward for each state (in consideration of uncertainties in the future), resulting in expensive computations due to the "curse of dimensionality"[15].

2.3 Value-based and Policy-based approximation

Due to the unfeasibility of computing the real value for $V_\pi(s)$, we need to use different approximations to reduce the computational expenditure. There are two main ways to use approximations in the MDP to achieve this goal.
The first one is the value-based approximation, where we exchange the value function $V_\pi(s_t)$ with an approximator function $\tilde{V}_\pi(s_t)$ in Equation (2). The resulting Equation (4) for an action in a state is often abbreviated with the Q-value, standing for the Quality of a certain state-action pair.

$$Q(s_t, a_t) = \max_{a_t} \mathbb{E}_\pi[r_t + \gamma \tilde{V}_\pi(s_{t+1})] \tag{4}$$

This further leads to the adaptation of the optimal action Equation(2), leading to the following representation:

$$\tilde{a}_t^* = arg\max_{a_t} Q(s_t, a_t) \tag{5}$$

The other way to use approximations in the MDP is to approximate the policy:

$$\tilde{a}_t^* = \tilde{\pi}(s_t, \phi) \tag{6}$$

By approximating the policy and not the Value-Function, we can derive an optimal action $\tilde{a}_t^*$ from just the state s_t once ϕ is set, without needing to deal with the recursive properties of Equation (2).

2.4 Temporal Difference Learning

Temporal Difference Learning is a method to update our approximations, both value and policy-based, by collecting state-action-reward sequences, which are then used to adjust the approximations after going through these sequences. For Q-Learning, an RL algorithm, we update our Q-value via the following equation:

$$Q(s_t, a_t)_{new} = Q(s_t, a_t) + \alpha * TD\_error \tag{7}$$

We transform our Q-value by adding TD_error, which represents the difference between the present Q-value and the TD_target, multiplied by the learning rate α, which dictates how much the Q-value can be changed per iteration.

$$TD\_error = TD\_target - Q(s_t, a_t) \tag{8}$$

We can split TD Learning into two methods. Off-policy means to adapt to the approximation independent of the action the policy would have taken in a state. For example, always updating our TD_target based on the best possible action in the future is a off-policy TD-Learning approach.

$$TD\_target_{off} = r_{t+1} + \gamma \max_a Q(s_{t+1}, a) \tag{9}$$

The other approach is on-policy TD Learning, using the current policy to update our Q-values by adjusting the approximations dependent on the actions the current policy takes. For example, updating the TD_target with the action our policy would take in the next state.

$$TD\_target_{on} = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) \tag{10}$$

2.5 Model-based vs Model-free

RL algorithms can be separated into model-based and model-free approaches. The model-based approach utilizes a model of its environment, given or learned by our Learner. The advantages are that model-based algorithms can be more sample-efficient due to their ability to simulate their environment with their internal representation without interacting. Through this, these algorithms can play different Atari games with a fraction of the interactions of others[16].
The performance of model-based RL heavily depends on this internal representation. If the model does not accurately represent the actual environment, the Learner's decisions may be suboptimal.
Figure 4 displays a small overview of different RL algorithms to their respective approach.

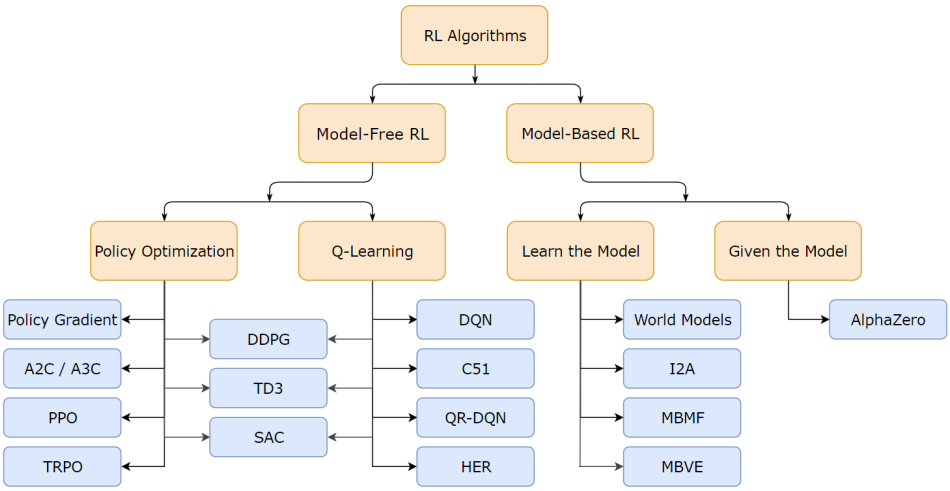


Figure 4: Different RL algorithms divided in model -based and -free approach[17]

On the other hand, there are model-free approaches, where agents are not creating a predictive model of how the environment works but directly learning from their interactions with the environment to determine the best actions to take.
This approach is particularly well-suited for situations where building an accurate model of the environment is challenging or computationally expensive. By not relying on a model makes them well-suited for situations where the dynamics of the environment change over time. The agent can continuously adapt its policy based on the feedback it receives from the environment without requiring it to update its underlying model update, which would bring further implications.

2.6 Offline vs Online

Offline and online RL describes how our learner gathers their experience in the environment. In offline RL[18], we use collected experiences(without the ability to interact with the environment during training) to train our Learners. One of the main problems is the quality of data and the spectrum of its state/action pairs it is representing. If these are not to a certain standard, we can't expect our Learners to act reliably and efficiently in this environment.
On the contrary, we have online RL, where our agent actively explores its environment to collect data, on which it is improving its behavior. This type of RL is well-suited for situations where continuous learning and adaptation are required.
While online training demands significant computational resources, its advantage over offline methods lies in its ability to adapt to the current system conditions, especially when there are substantial shifts in system characteristics before and after the start of the control process, as is the case in Non-Stationary environments.

2.7 Exploration versus Exploitation

Reinforcement Learning involves balancing the exploration and exploitation trade-off. During each step of an MDP, the agent faces a choice between taking an exploratory or an exploitative action.
An exploratory action is typically selected randomly from all available actions, following a uniform distribution. However, there are alternative exploration strategies, such as consistently choosing the least frequently selected action in the past. On the other hand, an exploitative action is the one that currently has the highest expected return.
The most commonly used approaches for determining the exploration-exploitation balance are greedy, ϵ-greedy, and simulated annealing.
Greedy action selection always opts for exploitation and completely disregards exploration.
In ϵ-greedy action selection, exploration is chosen with a probability of $0 < \epsilon < 1$, while an exploiting action a' is chosen with a probability of $1 - \epsilon$. Greedy actions aim to maximize immediate rewards, but exploration can potentially lead to a more refined understanding of the environment and ultimately greater rewards in the long run, which is the objective of ϵ-greedy actions. However, ϵ-greedy action selection never entirely stops exploring, even if the action values remain unchanged, due to the fixed percentage of exploratory actions. It can be formulated as follows[19]:

$$a_t = \begin{cases} argmax_a Q(s_t, a) \ if X > \epsilon \\ a' \ otherwise \end{cases} \tag{11}$$

Simulated annealing, on the other hand, also employs ϵ as the probability for selecting an exploratory action. However, the value of ϵ is not constant and can vary over time. It begins with a larger value, sometimes even 1, and then progressively decreases over time, often following an exponential decay[20].
The simulated annealing approach resembles the way humans intuitively learn, as we rely on trial and error until we develop a sense of favorable and unfavorable actions. Once we have acquired sufficient knowledge through exploration, we shift towards exploitation and apply our accumulated experience in practical situations.

2.8 Multi Agent Reinforcement Learning

Multi-Agent-Reinforcement-Learning (MARL) is an area of reinforcement learning where multiple agents interact with each other in a shared environment while learning to make decisions to achieve individual or collective objectives. With the introduction of interaction between multiple agents, the base problem increases in complexity, and new problems arise.
The agents can be controlled by one network, leading to a homogeneous set of actors[8], which all act the same if in the same state, or they are controlled by multiple networks, leading to different behaviors from each other. Additionally, the agents can be in a cooperative[8] or competitive setting like the Game of GO or Texas Hold'em[5].
New challenges that arise with the added interaction include coordination problems and Non-Stationarity due to the evolving strategies of other agents. This lack of knowledge of what another agent will do and suboptimal Nash-Equilibria are the core problems of MARL.
In the real world, a lot of the tasks are not solved by a single person or actor but instead by a pipeline of multiple groups with cooperating partners. That is why MARL is vital, as it enables different systems to solve complex tasks with multiple actors in dynamic environments by learning efficient strategies. There are many application areas for MARL, which include autonomous scheduling of multiple vehicles, multiplayer games, and many more.[21]

2.9 Baseline Algorithms

All Algorithms in this work are model-free, online, and homogeneous over all agents, meaning all agents are controlled by the same Algorithm. There will be three baseline algorithms, Deep-Double-Dueling-Q-Learning, Proximal Policy Optimization, and Advantage-Actor-Critic.

2.9.1 Deep Double Dueling Q-Network

The Deep-Double-Dueling-Q-Network (DDDQN) is a value-based algorithm, which consists of a Deep Q-Network (DQN)[22] improved with a Dueling Network Architecture[23] and Double Q-Learning[24].
In a DQN, we use an artificial neural network(displayed in Figure 5) that approximates the Q-values. It takes the current state as input and outputs Q-values for all possible actions. The DQN is trained to minimize the temporal difference error.
The Double Q-Learning aims to decrease the overestimating bias and the sensitivity to noise of DQNs and Q-Learning in general. This works by using two individual Q-networks, Q_1 and Q_2, trained by alternating between each other in training and evaluating each other. Leading to one of the networks selecting the best action and the other one evaluating the selected action.
To implement this, the standard TD-target is replaced with the following two functions where we choose one with an equal probability at each Q-value update:

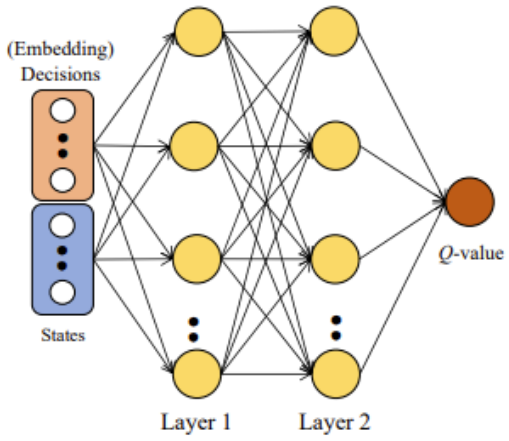


Figure 5: ANN to approximate Q-values [3]

$$TD\_target_1 = r_{t+1} + \gamma Q_1(s_{t+1}, \underset{a}{argmax}\, Q_2(s_{t+1}, a)) \tag{12}$$

$$TD\_target_2 = r_{t+1} + \gamma Q_2(s_{t+1}, \underset{a}{argmax}\, Q_1(s_{t+1}, a)) \tag{13}$$

The dueling network architecture(Figure 6) is also a technique to adapt the estimation of the Q-values. The dueling architecture allows our network to individually model the value of the state and the advantages of different actions, which can lead to more efficient learning due to the more detailed perspective on the state-action pairs.
Instead of approximating the Q-values directly, the dueling architecture splits the Q-value into the value of the state itself and the normalized advantages of each possible action in this state. Both these values then get combined again, leaving us with the following formula for the Q-value:

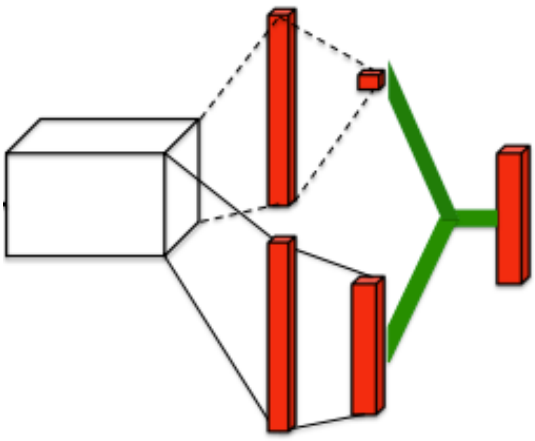


Figure 6: Dueling Network Architecture [23]

$$Q(s,a) = A(s,a) + V(s) \tag{14}$$

2.9.2 Proximal Policy Optimization

PPO[25], introduced by OpenAI in 2017, is a policy optimization algorithm known for its relative ease of implementation and stable training.
In PPO following equation is maximized:

$$L_{\text{CLIP+VF+S}}(\theta) = \mathbb{E}^{\pi_\theta}\left[L_{\text{CLIP}}(\theta) - c_1 L_{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t)\right] \tag{15}$$

This equation consists of three parts. $S[\pi_\theta](s_t)$ represents the entropy term that ensures that we can encourage exploration if wanted. $L_{\text{VF}}(\theta)$ is the value function loss term, which is the mean-squared-loss between the predicted value $V_\theta(s_t)$ of the state and V_t^{target}, being the expirienced value of the state:

$$L_{\text{VF}}(\theta) = \left(V_\theta(s_t) - V_t^{target}\right)^2 \tag{16}$$

$L_{\text{CLIP}}(\theta)$ represents the clipped surrogate objective function, it ensures that policy updates are performed within a "trust region" to prevent drastic policy changes compromising between utilizing new information for policy improvement and preventing excessive updates that may lead to instability. This is achieved by introducing a clipping mechanism in the objective function, constraining policy updates to a certain range, adjustable with ϵ.

$$L_{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \tag{17}$$

PPO enhances sample efficiency by reusing collected experiences across multiple iterations, leveraging each experience multiple times for learning.
Additionally, PPO has demonstrated effectiveness in a variety of tasks, providing a robust and versatile solution for training policies in reinforcement learning settings.

2.9.3 Advantage Actor Critic

The Advantage Actor-Critic (A2C) algorithm [26] integrates elements from both policy-based and value-based methods to enhance the training of an agent.
The fundamental Actor-Critic algorithm involves the simultaneous update of two components: the actor, which learns a policy for selecting actions, and the critic, which estimates the value of states or state-action pairs. The Actor-Critic framework benefits from an improvement loop, where the actor leverages the critic's value estimates to guide policy updates, as illustrated in Figure 7.

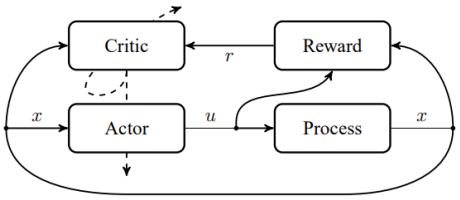


Figure 7: Actor Critic Architecture[26]

The A2C improves upon this by changing the critic. Rather than the critic using the Q-value to evaluate a taken action of the actor, it utilizes the advantage value (19), which returns the value of the action relative to each other in a certain state. This decreases the variance of the policy and stabilizes the model overall.

$$A(s,a) = Q(s,a) - V(s) \tag{18}$$

3 Related Work

3.1 Continual/Lifelong Reinforcement Learning

Lifelong Reinforcement Learning or Continual Reinforcement Learning refers to the idea of training a reinforcement learning agent to continually learn and adapt over time, facing multiple tasks/environments.
Traditional RL often assumes stationarity, meaning that the agent trains for a specific task and environment, which do not change over time. It might struggle when faced with new, unseen tasks that get introduced during its lifetime and are possibly not related to any tasks before.
Lifelong RL aims to address this limitation by enabling an agent to accumulate knowledge and skills over its entire lifetime, ideally allowing it to adapt to new tasks without forgetting what it has previously learned.
The goal of lifelong reinforcement learning is to create agents that can continually learn and adapt to a variety of tasks, making them more versatile and capable of dealing with Non-Stationary circumstances. This versatility is crucial for real-life deployment, where a lot of factors interplay and have unforeseen effects on the environment to which the agents need to react reliably.
There are many approaches to deal with Non-Stationarity in the RL context. In the following, we will present some of these approaches to gain insight into how Non-Stationarity can be addressed. In Figure 8, the main approaches are displayed. It is important to mention multiple methods can be utilized simultaneously.

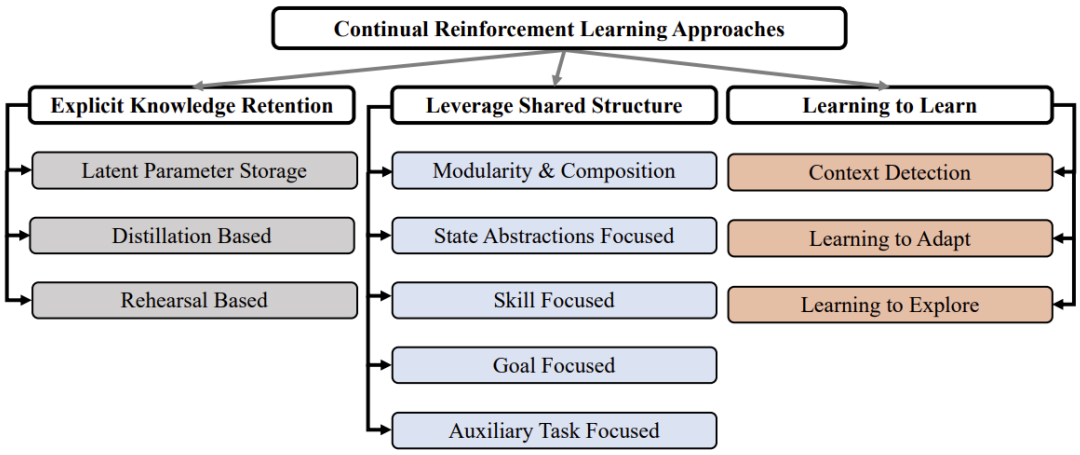


Figure 8: Taxonomy of Continual RL Approaches[4]

Explicit Knowledge Retention designates the techniques that store or revise past tasks to prevent catastrophic forgetting. This is done by training multiple networks for different tasks, preserving past knowledge by using older network configurations to guide the present learning process and rehearsing older experiences via a replay buffer.
The techniques presented in the Leverage Shared Structures section illustrate how our agent comprehends the structure of similar problems, allowing it to apply skills acquired in one scenario in another. This human skill of comprehending the inner workings of a solution and then utilizing this understanding in novel problems would benefit RL algorithms to act in continual environments.
With Learning-to-learn, also known as Meta-learning, models are trained to acquire the skill to learn new tasks efficiently. The models achieve this by using context detection, where the model learns to distinguish between different tasks, altering how it adapts to new tasks based on past experiences and learning how to efficiently explore new settings.
We can use multiple techniques unique to Non-Stationarity caused by changing interactions

between agents in MARL settings. A common approach is a centralized critic[27] or using communication between agents. We can roughly split them into two concepts. Agents that communicate with shared or connected neural networks[28] and agents utilizing communication channels to send messages between each other to solve tasks[29].
The mentioned ways are a small selection of algorithms and techniques to adapt to an ever-changing environment. For a more in-depth look, we refer to the following papers[6][30][31].

3.2 Curriculum Learning in Machine Learning

Both humans and animals exhibit improved learning outcomes when the examples are arranged in a coherent sequence, progressively introducing more concepts and increasing complexity rather than being presented with randomly ordered or too complex data at the start. In the context of machine learning, this ordering of training data is called Curriculum Learning[33].
One of the earliest instances of a curriculum in the ML context was in the paper [34] from the year 1985 about the cart-pole-balancing problem, where a pole, stuck with one side to a cart, needs to be balanced by an algorithm, which is able to move the cart below it, as seen in Figure 9. In the paper, it was called "directed training" instead of CL and led to the task of balancing a short pole being acquired with fewer failures by beginning to train with a larger pole and reducing its size incrementally.

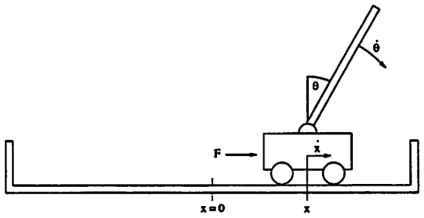


Figure 9: The Cart-Pole Problem[32]

Today, there are a lot of different methods to implement a curriculum which find application in many fields like medical imaging[12], speech processing[35], navigation tasks [36], and many more[37] leading to improved sample efficiency and better convergence of ML Algorithms [33].
By regarding the training process with CL as a continuation method of finding the optimal parameters to receive a wanted output from certain inputs, we can see earlier simpler training as a smoother objective, where it is easier to find the best behavior and by tracking this optimum through increasing complexity, the model performs better in the evaluation data. Figure 10 displays this type of tracking, we can see how the curriculum guides the model to the global optimum of the target objective. The same can be achieved when working with noisy datasets, where the learned capabilities of our algorithms can be improved by starting with cleaner parts of the dataset that have less noise and then introducing parts which contain more noise[7].

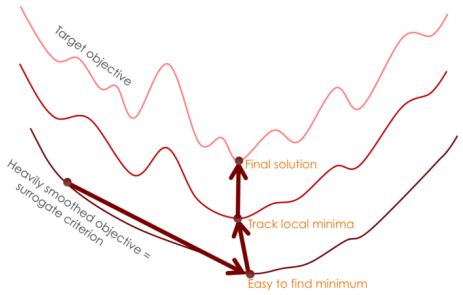


Figure 10: Schematic Display of Tacking the global optimum via CL.[7]

Viewing this problem in this manner also brings to mind how different parts of the curriculum can push the behavior of our model in a wrong direction and, therefore, possibly lock the model in a suboptimal part of the parameter space. This is particularly the case if we employ unideal custom environments that are not part of the evaluation data.
With CL improving the results of different ML models, it is a good fit in combination with RL. Due to the nature of RL, the agent often gets stuck in suboptimal behavior, which could have been prevented by guiding the agent in the direction of the global optimum. This guidance can be achieved by deploying a curriculum with environments containing a more limited state/action space than the target task, facilitating the algorithms to differentiate between valuable and invaluable inputs[38].

3.3 Creating a Curriculum for Learning

With CL improving the capabilities of RL, it is important to think about what needs to be considered when constructing a curriculum. When creating a curriculum, we mainly need to pay attention to three parts: task generation, sequencing, and transfer learning. If neglected, the curriculum is prone to be inefficient or useless.

3.3.1 Task Generation

Task Generation refers to the process of creating a set of tasks or subtasks that a model needs to learn progressively. These tasks should be designed in a way that starts with simpler concepts and gradually increases in complexity. So that their knowledge transfer through them is beneficial and carries no negative transfer, meaning that the task leads to a declining performance in the evaluation environment[38].
There are two main ways to generate useful tasks. On the one hand, we can just utilize easier versions of the target task and increase the difficulty, and on the other hand, we can split the Problem, in our case, the TSP, into smaller tasks and teach our agent these subtasks in succession.
Most methods operate under the premise that the domain is definable with multiple parameters, and varying values of these lead to the output of separate tasks[39]. These environments can be handmade[40] or automatically generated[41].

3.3.2 Sequencing

Sequencing involves determining the order in which these tasks are presented to the model learning the task. The tasks need to be introduced in a meaningful and gradual manner. Otherwise, our model may be stuck in local optima due to the introduction of concepts in a suboptimal order, leading to catastrophic forgetting[42].
One prominent idea is the use of Teacher-Student Curriculum Learning[43], where we have a teacher network choosing the environments in which our student network is learning. The teacher network tries to maximize the learning effects of the student network by selecting the right environments.
Overall, the idea of sequencing is to build on previously acquired knowledge and facilitate the learning experience for the agent[38].
For example, it would be harder to impossible for somebody learning to write before learning to read. This sequencing allows our model to learn crucial skills first and then build upon them to improve in the overall task.

3.3.3 Transfer Learning

Transfer Learning comes into play as the model advances through the curriculum. It should maintain and use the abilities learned from previous tasks to improve its performance on subsequent tasks. This knowledge transfer enhances the model's ability to generalize and adjust to new challenges in an efficient manner.
Without this transfer of learned abilities across different environments, we would need to learn every subtask from the ground up, which needs much computing power without improving the performance of the network on the final task, rendering the curriculum useless.
With good knowledge transfer in a MARL setting, we are able to start training with a small number of agents and then gradually increase the number of agents while being able to use the experiences collected in the simpler case to act more efficiently later in more complex environments with more agents[11].

3.4 The Train Scheduling Problem

In the long-existing ecosystem of train logistics, many processes still rely on outdated suboptimal solutions that do not utilize the technology of today to reduce cost, improve customer and worker experience, and ameliorate safety.
The TSP is one of these problems. It involves optimizing schedules for multiple trains on a rail network and rescheduling them if needed. Its complexity can vary from relatively simple problems for smaller networks to highly complex optimization challenges for large and congested rail systems.
We can deconstruct the TSP into a graph problem, in which we need to find different paths for trains through the network of tracks without them intersecting in a location at the same time. For example, in Figure 11, you can see a relatively simple train network (implemented in Flatland) with one train in each of the stations that need to reach their goal on the other side. This train network can be converted to the graphs in Figures 12 and 13. In Figure 12, you see poorly chosen paths for both trains as a result of these unoptimized routes, they are crashing together in the middle node at time step three. In contrast to this are the paths of Figure 13. Here, we route the trains past each other without their time-location-position overlapping, and therefore, this solution enables both of our agents to reach their goal.

Figure 11: Simple TSP-Problem

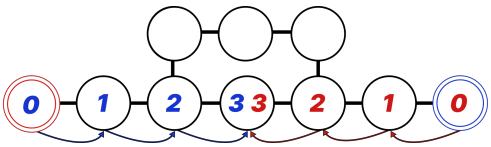


Figure 12: Invalid Solution of the TSP

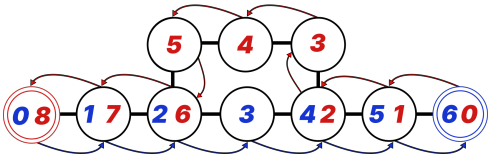


Figure 13: Valid Solution of the TSP

Efficient train scheduling is essential for ensuring safe and reliable rail transportation while minimizing delays, maximizing throughput, and balancing the load over all tracks. It is an integral part of both passenger and freight rail systems.
We can subdivide the TSP into the Multi-Agent Pathfinding(MAPF) Problem and the Vehicle Rescheduling Problem(VRSP).

3.4.1 Multi-Agent Pathfinding

Single-agent pathfinding is a standard computational problem where the objective is to find the optimal path from a starting point to a goal location for a single entity. The goal is to minimize the cost or distance traveled while avoiding obstacles and reaching the destination. This problem is expanded to the MAPF problem, in which we need to find routes for multiple agents but also need to consider them as obstacles for others.
Solving the MAPF problem by having each agent manage it sequentially is proven to be NP-hard[44], which requires the solutions to utilize different techniques to minimize computational work while upholding performance.
The MAPF also finds usage in many more logistics problems like aviation managment[45], robot organisation[46], and many more, which does not mean that it is a good idea to use the same solution for all of them due to a lot of domain-specific characteristics[47] which further complicate the problem in one direction or another.
In the train scheduling context, we need to consider that the transitions of switches are dependent on the travel direction of a train and that some trains can not change their travel direction without the help of looping railway structures or other means of support.

3.4.2 The Vehicle Rescheduling Problem

The VRSP describes the needed capability of our trains to react to unforeseen events that restrict the previously planned progress of single or multiple trains and thus call for the ability to dynamically reschedule.
Learning how to deal with this problem would benefit many transportation problems like mail delivery, ambulance routing, and public transport. These jobs need to adapt their course constantly, depending on traffic jams, road closures, or the change of their destination[48].
Dynamic rescheduling is essential in the TSP because it allows the system to adapt to unforeseen disruptions such as delays, breakdowns, or changes in demand. By incorporating dynamic rescheduling, the system enhances overall reliability and customer satisfaction in the face of unpredictable events.

3.4.3 Other Railway Application for Machine Learning

Aside from the TSP, there are a lot of other problems in the ecosystem of railway systems that could profit from Machine Learning and specifically from RL. In the following, we will present some areas that could be improved with the help of ML. Most of them are actively researched and improved constantly.
Still somewhat close to the TSP is the problem of train shunting. The train unit shunting problem is a specialized optimization problem that deals with the efficient movement and rearrangement of train parts in a rail yard. The goal of the train shunting problem is to organize the train parts in a way that ensures that they are placed properly for transport, assembling, cleaning, and other jobs[49][50].
Railway safety at train stations can also profit from Machine Learning. For example, we can use CCTV cameras to observe stations and automatically detect people via image processing in risky situations or unallowed areas and act upon these recognitions of dangerous settings[51]. Moreover, we can improve safety systems by predicting characteristics of individuals at risk and using these predictions to protect these people[52].
Another big area is Railway Inspection and Maintenance [53]. This area still heavily relies on human assessment, which is not always able to detect all signs of deteriorating rails and trains, especially if these deteriorations are in their early stages. Computer vision can support or even replace inspectors in finding weaknesses or parts that require renewal and, therefore, make the rails and trains traveling on them more reliable and safe.

3.5 Operations Research

Operations Research (OR) is its own broad field of research using applied mathematics and science that focuses on solving complex decision-making problems. OR is used as a solution in a wide range of applications. These include logistics, supply chain management, finance, scheduling, and many more.
OR uses mathematical models of the environment, analytical methods, and other computational techniques to optimize the decision-making process. The discipline's strength lies in its ability to solve patterns within data and make strategic decisions. Through the application of sophisticated algorithms and computational approaches, OR plays a pivotal role in addressing multifaceted challenges, ultimately contributing to the improvement of overall system performance and effectiveness.
One problem of OR is that it typically scales nonlinearly with the size of the problem, leading to a high computational workload with growing environments or complexity and therefore making it an unsuited solution for big and complex problems (how they often appear in real-world scenarios) when not having access to a lot of computing power or needing to sacrifice the accuracy of the solution against computing power by implementing approximations.

The second issue centers on the critical need for a sufficient deconstruction of the problem, establishing a solid groundwork for further development. Without an adequate representation of the problem, we are not able to build efficient algorithms.
With its roots in military organization and resource allocation in the Second World War [54], with the railway network being a crucial part of military operations, OR solutions were used relatively early in freight transport and later in passenger transport. In this way, OR is used as a solution to a lot of organizational tasks and logistics problems today.
Even though we are not building any OR algorithms in this thesis, it is always important to incorporate the way of thinking that is required to build OR solutions. Deconstructing real-life problems into mathematical models helps us comprehend the problem on a much deeper level, enabling us to find answers that otherwise would have stayed hidden from us. This way of thinking helps us, especially when building curricula for RL algorithms.

3.6 Flatland Challenge

The Flatland Challenge is a competition that invites anyone to join and submit their solutions to contest for the top spot by accumulating the highest number of points. This challenge unfolds on a two dimensional railway grid, where trains must navigate from their starting points to their designated destinations. Participants earn points for successfully guiding trains to their targets and for achieving completion of an environment, signifying that all trains within the scenario have reached their respective destinations. The complexity of these environments increases progressively with the expansion of grids and the inclusion of more trains, making the challenge more demanding[55].
This challenge was brought to life in the year 2019 by the Deutsche Bahn, the Société Nationale des chemins, the Swiss Federal Railways, and AI-Crowd to encourage more solutions in the TSP, especially in combination with RL [56].
Despite one of the primary objectives of the Flatland challenge being the advancement of RL in railway scheduling, it has become evident that RL methods presently struggle to match the effectiveness of OR or approaches that combine both, as indicated in the outcomes from the 2019 Flatland challenge[57]. As a consequence, the organizers of the event split the solutions from there on into an "Overall-Track" where all solutions could participate independently of their approach, and an "RL-Track" where exclusively RL approaches could enter.
At the time of writing this thesis, the last Flatland Challenge took place in 2021.

3.6.1 Operations Research Solutions

In 2019, the winner created a solution revolving around planning deadlock-free paths in a state-based system, with the state consisting of location and time. Planning occurs dynamically because random malfunctions occur during simulation, necessitating path re-computation, which can lead to deadlocks. But if the visiting order of the agents is maintained for all agents, future deadlock avoidance would be guaranteed. So the winner implemented a strategy that always kept the visiting order the same in the case of a malfunction to prevent deadlocks and improved the new routes of the trains after the conflict adaption.[58].
In 2020 and 2021 "An_Old_Driver" won both competitions[59][60]. The solution employed a MAPF algorithm that prioritized trains and scheduled them in descending order of priority. It aimed to calculate collision-free paths while avoiding conflicts with already scheduled trains. Malfunctions were also addressed during execution. Conflict resolution involved partial re-planning, consisting of checking all following switches/intersections of the affected train and reordering the sequence of the trains driving through if needed[61].

3.6.2 Reinforcement Learning Solutions

In the RL-Track, there were many different approaches to solving the train scheduling problem with RL.
A big differentiator between the solutions was the selection of the algorithm and training itself. A lot of answers to the problem were different Algorithms like Deep Q Networks, Proximal Policy Optimization, Actor-Critic, and Imitation Learning.
The collection of data on the environment was another key component to success in this competition. It is important to collect enough data so our agent can act reliably on the rail network, but not too much due to the threat of collecting useless information, which adds noise and complicates the learning process of the agent as a result. All successful RL solutions used some kind of tree observation, where only the subsequent rails are scanned to a certain depth.
Another characteristic that set apart the different solutions was the adaptation of the reward function. The reward function determines the behavior of our agents, so adjustments to it can lead to crucial improvements in learning speed and performance. Every high-performing competitor edited the reward function to punish unwanted behavior like ending in deadlocks, incentivize wanted behavior like reaching its destination fast, or both.
Many competitors also used unique approaches to improve performance like enabling the trains to communicate with each other to coordinate with each other, implementing a priority value to solve conflicts, postponing the spawn time of trains, and many more. Many algorithms utilized the ability to postpone the spawn of a train, leading to less dense traffic but worse arrival times, but overall the scores improved due to fewer blockages, as a result of a less cluttered environment.

4 The Flatland Simulator

We are using the Flatland-Simulator[8] for this bachelor thesis. Created by AIcrowd in cooperation with different European railway corporations, designed to work on the major question: "How to efficiently manage dense traffic on complex railway networks?"[56].
Flatland abstracts real-world railway systems into a two-dimensional grid world. Flatland offers a customizable and extensible environment suitable for studying and solving train scheduling problems, making it a valuable tool for research and development transportation management in combination with RL.

4.1 Creating the Environment

In Flatland, every environment lies on a two-dimensional grid in which every cell can be one of eight types displayed in Figure 14. Each cell can be mirrored and turned by 90 degrees and needs to connect to a valid cell to prevent tracks from running into empty cells. If these cells get rendered, they look like in Figure 15. On each cell with train tracks, we can place a train station(destination of a train), the starting location of a train, or both.
Through these, we can display a wide array of different rail networks. We can subdivide them into Random Environments and Custom Environments.

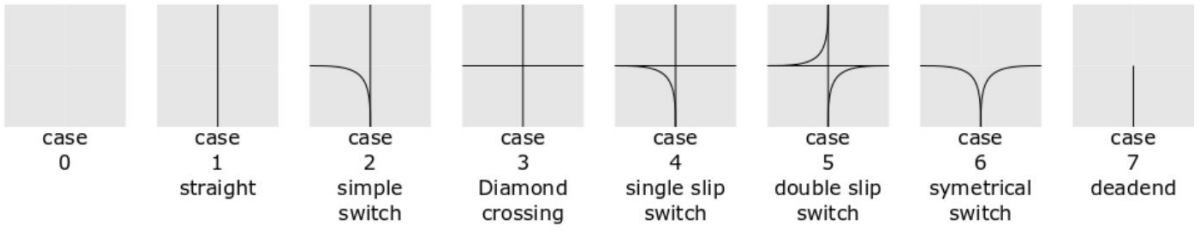


Figure 14: The eight type of cells [8]

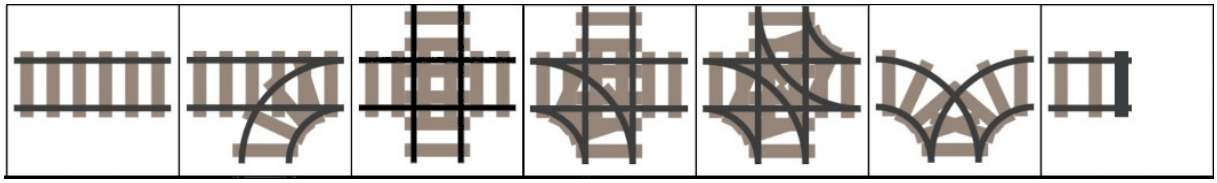


Figure 15: The seven rail cells stylized [62]

4.1.1 Random-Environments

We need many Environments, with a broad spectrum of railway networks, to train our agents with reinforcement learning. Otherwise, our model could exploit certain features in its training world instead of being able to adapt to new rail constellations by learning a solution to the task. This problem is called overfitting.
To prevent this, we use randomly generated environments via Rail-Generators in the simulator. Flatland provides us with the sparse_generator rendered on the left in Figure 16 . It generates rail networks that resemble internally good connected cities and longer less connected paths between these cities.

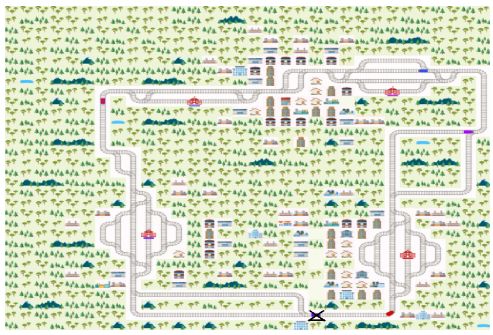


Figure 16: Random-generated world

4.1.2 Custom Environments

On the other hand, we need custom environments for two different reasons. First, we can build unique rail networks. These have not been seen in the training phase of our RL model, so they are an interesting way to test its adaptability to unknown situations.
Second, we can create simplified models of real-world railway networks to test the usefulness of different models in real-world scenarios like the rail network shown in Figure 32 used for evaluation.
Third, these custom environments are important to build your environments, which enables us to build complex curricula by using these environments as building blocks. We can implement some parameters into the custom environments to randomize or adjust them, making them a mix between randomized and custom environments.

4.2 Agents

Each agent controls one train, and they have no way of communicating with each other. A train has a set of features, can observe its environment, and act upon these inputs with multiple actions. Each agent receives a reward dependent on these actions.
The different models update their networks by using the state-action pairs of all agents.

4.2.1 Features

Features are the Characteristics of each train. They are accessible at every timestep of the simulation by the train itself and other agents.
These Features are:

- Positions : start, target, current, old position
- Directions: start, current, old direction
- Moving: moving, stopped
- Handle : ID
- Malfunction: how long, how often
- Speed: speed value between 0 and 1

4.2.2 Actions

At each time step, our agent can decide on one of five actions if the action is valid. An Action is invalid if it would cause two agents to be on the same cell at the next time step or if the train cannot perform the action because the possible transitions the cell provides do not match with the action the agent chose. When the action chosen isn't valid, the DO_NOTHING operator gets executed.
The five actions are as follows:

- DO_NOTHING : Repeat the action of the last time step or STOP_MOVING
- MOVE_LEFT : Make a left turn
- MOVE_RIGHT : Make a right turn
- MOVE_FORWARD: Travel forward or reverse direction on the dead-end
- STOP_MOVING : Stay on current cell

4.2.3 Observations

Observations are the view of our agent on its environment. These observations are a significant part of our agent. In Flatland, there are three usable presets displayed in Figure 17. These vary in efficiency, with some offering an efficient way to collect the information essential for our Learners.
We can also create new observations from scratch or combine multiple models to have an efficient view of our environment. Preferably, observations should only give the agent all the crucial information it needs to take the right actions, without data it can't utilize.

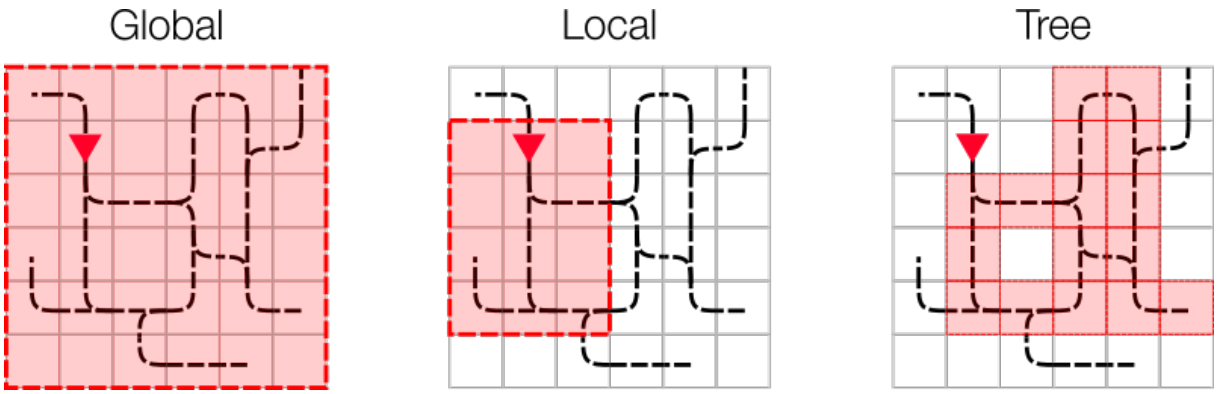


Figure 17: The three observation presets in Flatland[63]

In **Global Observation**, every agent has a full view of the environment. Every agent gets the position, target, direction, speed of himself and others, the malfunction, and a transition map displaying the possible movement given a direction on a cell. Global observation supplies the agent with information of varying significance. With this preset valuable information might get lost in the amount of data.
Local Observation is limited to a given height and width, to define the observed range around the train. This reduces the quantity of unimportant data. The information the agent receives for each cell remains the same as in global observation.
Furthermore, the Local Observation introduces a distance map that returns the distance between each train-target pair. Otherwise, our agent wouldn't be able to know how close he is to his destination unless it's in his local range.
A common problem is the insignificance of cells behind the train and parallel tracks, which are unconnected to the rail of the observing train.
The **Tree Observation** takes advantage of the track network's graph structure. The observation is created by exploring a four-branched tree starting from the agent. Each branch follows the licit movements until it encounters a cell with multiple legitimate actions, called a switch. The information collected along the track is stored as a node in the tree. This includes:

- distance to target cells, agents, blocked switches and next node
- conflict predictor
- minimum remaining distance to target if path is chosen
- number of agents ready to depart, going in the same and opposite direction
- malfunctioning agents and how long they will stay that way
- slowest speed of agent in same direction

The advantage of this type of observation is that we only use data on the traveling path of our agents, which leads to efficient information collection. We use the basic Tree Observation with a depth of three for all algorithms.

4.3 The Reward Function

In Flatland, each agent can receive a reward r_a at each timestep. These rewards are between 0, acting the best way possible, and -1, performing the worst way possible.
With the provided reward function 19, we have a sparse reward setting in Flatland, meaning our agent receives few rewards. This provided reward function gives each agent i a score $a_{\text{shortest path}}(t)$ dependent on its distance to its goal at the end of an episode at timestep T. The sparse reward setting introduces challenges because our agents get delayed feedback on their actions and a decreased sample efficiency due to the increased quantity needed to explore the environment.

$$r_a = \begin{cases} r_a = a_{\text{shortest path}}(t) & \text{if } t = T \\ r_a = 0 & \text{else} \end{cases} \tag{19}$$

4.4 Timetables and Delays

4.4.1 Schedules

During initialization, each train is assigned both the earliest departure time and the latest arrival time as attributes. By default, the earliest departure time is selected randomly via a uniform distribution between 0 and the $departure_{max}$.
With $departure_{max}$ being:

$$departure_{max} = T - t_{shortestpath} \tag{20}$$

With T being the last episode and $t_{shortestpath}$ describing the minimum time an agent needs to travel from its destination to its target without the influence of other trains or Malfunctions. In most cases the agents are able to reach their target before the end of the episode.

4.4.2 Malfunction

To simulate unforeseen events that restrict the ability of a train to move, like engine failures, medical emergencies on the train etc., we use Malfunctions. In Flatland, we define the probability of the trains enduring a Malfunction at each timestep and a time range specifying how long the Malfunction lasts. During the Malfunction, the trains can't move and block the track for all other trains during this time.
By adding Malfunctions, we introduce the rescheduling problem into our simulation. Without it, our agents could plan their whole path in cooperation with each other from start to finish and would not need to adapt to anything. A malfunction forces our agents to actively replan during travel to avoid crashes and suboptimal times.

5 Methodology

5.1 Overview

This section is split into three subsections, with the first two dedicated to constructing a curriculum from the sub-problems of the TSP, the approach is displayed in Figure 18.
First, we deconstruct the TSP into multiple smaller sub-problems, which helps us understand the underlying dynamics on a deeper level and gives us a foundation to build a curriculum on. These subproblems consist of: pathfinding, the problem of an agent finding a route from its current position to its destination; malfunctions, the problem of unforeseen stoppages of other trains; deadlocks, the problem of one or more trains ending up in a constellation that leads to a permanent halt of the involved trains; and train-speed-differences, the hindrance of one or more trains to travel at their full speed as a result of a slower train in front of them.
Additionally, the increased interaction between these sub-problems in bigger environments with higher agent counts makes the TSP more complex than the sum of its parts, requiring the models to not only learn the skills to solve the sub-problems isolated but also to be able to use these skills jointly to act reliable in a before unseen constellation of problems.
Second, we introduce four different curricula for training. The first curriculum is the No Curricula, which represents the common training approach of training the model in the evaluation environment without any changes. The second curriculum is the Simple Curriculum, which starts off with simpler variations of the evaluation environments and increases in complexity during training till reaching the completeness of the evaluation environment.
The final two curricula consist of three customized environments: pathfinding, malfunction/train-speed differences, and a deadlock environment. Each of these custom environments is derived from one or more sub-problems previously formulated in the preceding section. One of the curricula is with rehearsal, and the other is without, meaning that one of the curricula rehearses environments in intervals.
The last subsection covers three distinct techniques aimed at addressing Non-Stationary environments in RL. These methods enable our baseline algorithms to preserve past knowledge, reduce catastrophic forgetting, and anticipatedly leverage the learning improvement the model gains from the custom curricula.
These methods consist of an adapted version of the upper confidence bound exploration that explores more when the model enters a new environment, the elastic weight consolidation, where we make it harder to alter weights/biases that are important to certain tasks, and the adaptive rational activation function for our neurons, which are neurons which adjust themself dependent on the input distribution during the training.

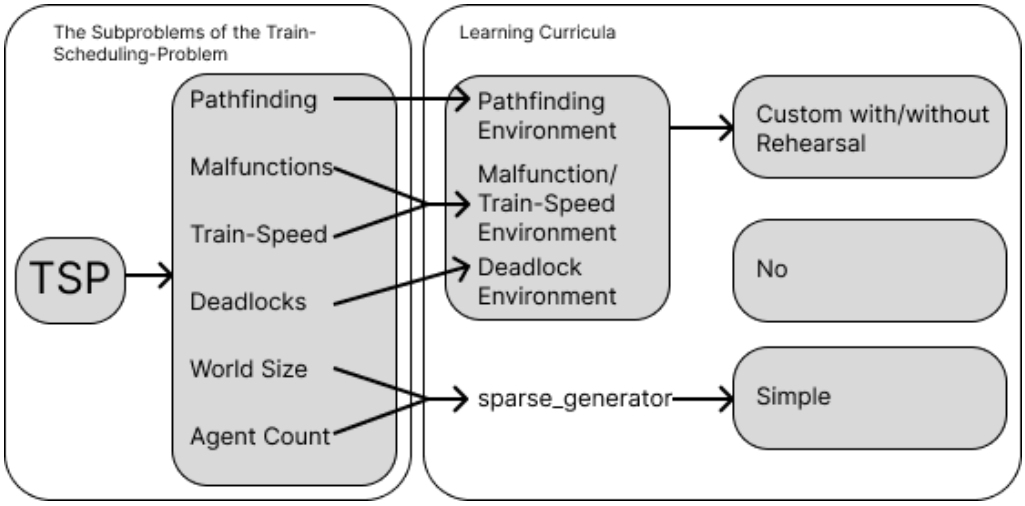


Figure 18: Approach of building a Custom Curricula derived from the sub-problems of the TSP

5.2 The Subproblems of the Train-Scheduling-Problem

In railway problems, we have two main metrics. On the one hand, we want every train to reach its destination. On the other hand, we want a train to be as fast as possible. To create a purposeful curriculum that enables our algorithms to excel in both of these metrics, we need to split the TSP into its subproblems.

5.2.1 Pathfinding

Efficient pathfinding is one of the primary parts of the TSP. If a single train can't reach its destination without taking unnecessary detours, it is destined to fail in an environment with other agents and uncertainties.
So it is crucial to train a single agent to find his way from the start to his destination through the train network otherwise, it is useless to improve other parts of the train scheduling problem if our agents do not have the foundation to reach their destination efficiently.
Problems that can arise if the pathfinding is not trained well enough are the possibility of an uncontrollable long way from start to goal, where the agent wanders through the network until it reaches its target.
The existence of random stops on tracks with no switches leads to suboptimal times because it has no impact on the overall state of the network and the interaction between trains if an agent stands still in a part where it otherwise is just able to move along one path.
The ultimate goal for pathfinding would be an agent always choosing the shortest path, wasting no time with unnecessary turns or stops.

5.2.2 Malfunctions

As explained in Chapter 4.4.2, malfunctions can appear randomly and last a certain amount of time, causing different scenarios where our agents need to adapt to the situation.
There are two main cases where the malfunction causes a problem requiring the agents to act or even make foresighted decisions, like not driving behind another train anticipating the malfunction.

Same Direction

If multiple trains drive behind each other in the same direction, the malfunction of a train will affect all of the other trains behind it, increasing their travel time.
In Figure 19 this scenario is depicted with two trains. The front train causes the train behind it to lose time even though it has no malfunction. The back train could have chosen the longer bottom way to have a better time in the worst case of a malfunction.

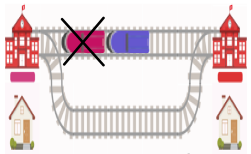


Figure 19: Purple train effected by the Malfunction of the pink train

Contrary Direction

Malfunctions often can lead to a Deadlock or a longer way for one of the agents, especially if the trains move in contrary directions and they are not adapting to the circumstances.
In Figure 20, you can see a case displayed where a malfunction of the pink train leads to a deadlock for both agents if the red agent continues on its path without stopping. Both can reach their destination without problems if the purple train waits.

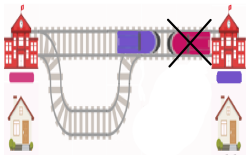


Figure 20: Deadlock caused by a Malfunction

5.2.3 Deadlocks

Deadlocks are the most common reason for trains not reaching their goal. A deadlocked train can not move or escape a sequence of tracks. They should be avoided even if it increases the travel time of one or more trains since a delayed train is more acceptable than no train. Three types of deadlocks can occur if trains can't move backward, like it's the case in the Flatland.

Common-Deadlock

In Flatland the most frequent deadlock is the Common-Deadlock with dangerous aftereffects in the real-world, like derailing and damage to the train and its surroundings.
In Figure 21 there are two trains on the same track but in opposing directions, leading to a deadlock of two trains not reaching their destinations.
This deadlock is critical to consider due to its simple nature and the consequence of two trains not getting to their destination. An unpleasant side effect is the blockage of the path where the deadlock occurred, making it unusable for all other trains.

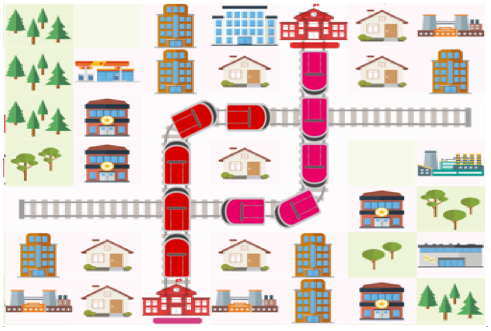


Figure 21: Common Deadlock

Cyclic Deadlock

The Cyclic-deadlock happens when there is a cyclic dependency between two or more trains to free a track while simultaneously blocking the track of another train needed to proceed. This is comparable to a deadlock in network theory, but we can't remove one of the involved actors.
Figure 22 shows the simplest case of such a deadlock. The pink trains cannot continue until the red trains free the track and vice versa.
This constellation is unlikely in real life and even less expected in Flatland as a result of a train length of 1 cell.

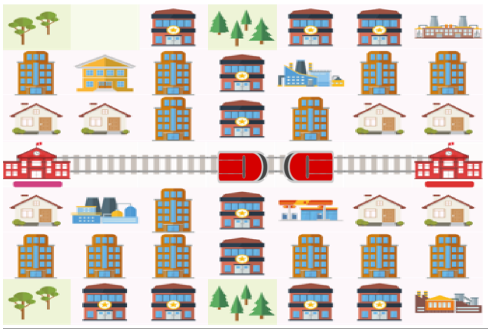


Figure 22: Cyclic Deadlock

Track-caused Deadlock

As its name suggests, the Track-caused Deadlock arises from the design of the train track itself. The train route enables the agent to access a part of the rail route which it cannot leave.
In Figure 23, the agent is trapped in a loop, which he can't escape due to the structure of the track.
This Deadlock is uncommon in real-life scenarios. These constellations of tracks only exist in shunting yards, where trains are slow and drive backward if needed. In Flatland, such structures do not appear in environments generated by the sparse_generator, but be build manually in custom environments.

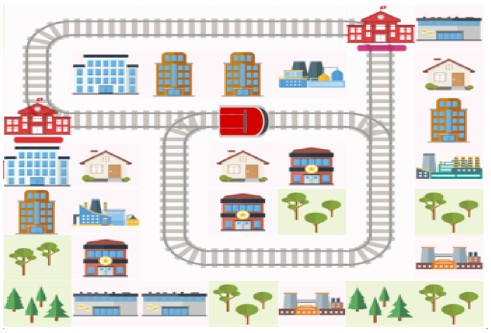


Figure 23: Track-caused Deadlock

5.2.4 Train-Speed-Difference

In Features 5.2.1, it is explained that each train can have a speed between 0 and 1. This trait leads to conflict between trains and worsens the overall travel time of the trains. Because of the Train-Speed-Difference, the agents can end up behind each other and slow down the trains behind them, that could travel with a greater velocity.
For example, in Figure 24, the pink train has a speed of 1 while the purple train has a speed of 0.2, which leads to the pink train not being able to drive at its full pace and being restricted to the velocity of its predecessor.

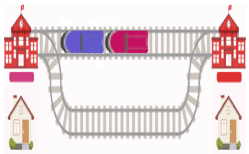


Figure 24: The purple train slows down the pink train

5.2.5 Combination

We need to keep in mind that these different problems do not always appear isolated but in a variety of combinations in all kinds of situations. In Flatland, it is common that different problems snowball into more, causing a chain reaction, blocking major parts of the railroad and slowing down the overall traffic.
Also, agents can not avoid all conflicts. In some cases, the agents have to choose the actions that have the least unfavorable consequences.
For Example, in Figure 25, you can see how a Malfunction of the purple train in a Contrary-Direction to the green train causes a Common-Deadlock and blocks the tracks as a result, leaving the red train in a track-caused-deadlock, not being able to reach its goal.
The green train could have prevented this situation by waiting for the purple malfunction to resolve before continuing toward its destination.

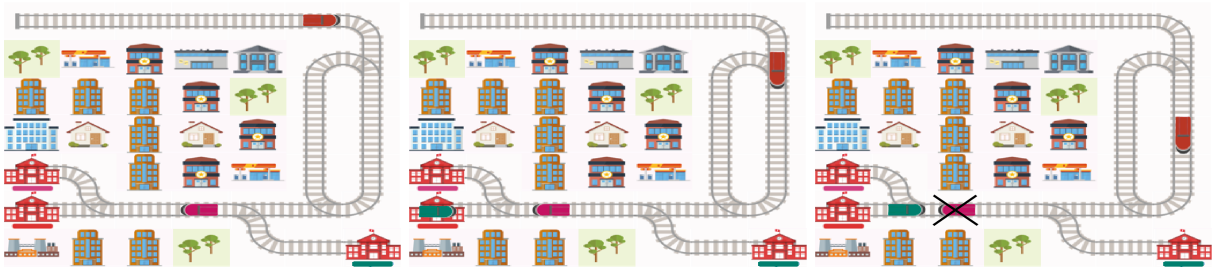


Figure 25: Chain-reaction of Problems

5.2.6 Environment-Size and Agent-Count

The complexity and the probability of conflict start increasing with expanding environments and a larger agent count.
The bigger the environment, the longer the ways the trains have to travel from A to B, exposing the agents to more decisions and possible conflicts.
With larger agent counts, the interaction between all of them becomes more and more complex with the increasing interplay between the trains, leading to more severe and longer-term consequences of unfavorable actions.

5.3 Learning Curricula

5.3.1 No Curriculum

The simplest type of training to implement is to utilize no curriculum at all. We train our network on the standard environment in which it should act reliably after the training. For the "No Curriuclum", we will use the sparse_generator to generate 64x64 environments with 14 agents, a malfunction rate of 1/1000 at each timestep, and trains with evenly distributed speeds between 1, 0.75, 0.5, and 0.25 for 1 million network updates, documented in 5.3.3 under "No Curriculum".
This approach is one of the most common because it is easy to realize due to the training and evaluation environment being the same, not needing any further work creating extra environments for training.
One of the main problems is that No Curriculum punishes agents by dropping them into a very complex environment without prior knowledge, not knowing which combinations of actions lead to favorable results and therefore needing more time to get a grasp of the behavior of the states in combination with different actions.

5.3.2 Simple Curriculum

In the Simple Curriculum, we start with simpler and smaller variants of the evaluating environment and increase its size and the number of agents incrementally until we reach the configurations of the evaluating environment. This approach is leading to a more forgiving environment in the beginning and adding complexity in a hopefully feasible manner to our agents.
To implement this curriculum, we are utilizing the *sparse_generator* and start with an environment size of 16 times 18 (the smallest possible environment created with the *sparse_generator*), with one agent, no malfunctions, and no train speed differences. After that, we increase the area of the environments, the agent count, the malfunction rate, and the train speed differences by increments until we reach the settings of the environment described in the No Curriculum. For the exact values, see 5.3.3 under Simple Curriculum. In Figure 26 such a curriculum is displayed.
The main advantage of the Simple Curriculum is the ease of implementation because we scale and tweak different parameters of the environment without creating our own environments.
On the other hand, this curriculum could promote overfitting and not help our algorithms acquire the skills to solve the TSP due to all environments being similar to each other.

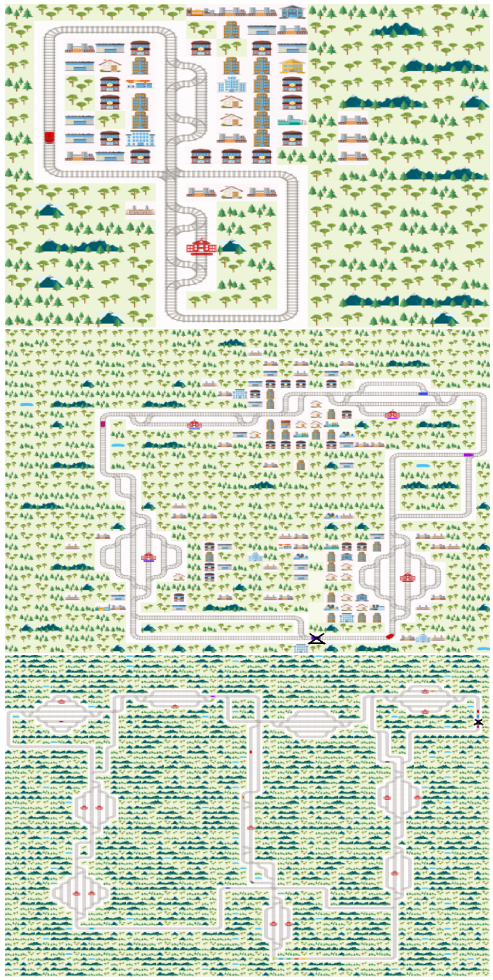


Figure 26: Graphic Display of the Simple Curriculum

5.3.3 Custom Curriculum

In our Custom Curriculum, we alternate between three different environment blueprints, each derived from the problems stated in 5.2 The Subproblems of the Train-Scheduling-Problem. Each of these environments teaches the agent one or more skills to solve the sub-problem and, therefore, act more efficiently in the overall task of train scheduling.
The advantage of the Custom Curriculum is that each environment is fitted to one of the named problems and teaches the model the skill to solve these tasks.
The disadvantage is the increased work in the creation of these environments and the multiple new problems that can arise, like bad Knowledge Transfer, the simultaneous application of these skills, and the introduction of environment Non-Stationarity.

Pathfinding Environment

The first task for our network is pathfinding. We start training in a small world the size of 8 by 8 with one agent. The size of the environment increases with each iteration while staying with an agent count of one.
We begin in such a small world to increase the probability of the agent reaching its goal and receiving a reward. We only have one agent to remove the risk of any problems, including two or more agents, and isolate the issue of pathfinding to its core.
Figure 27 displays the Pathfinding task in three different sizes.

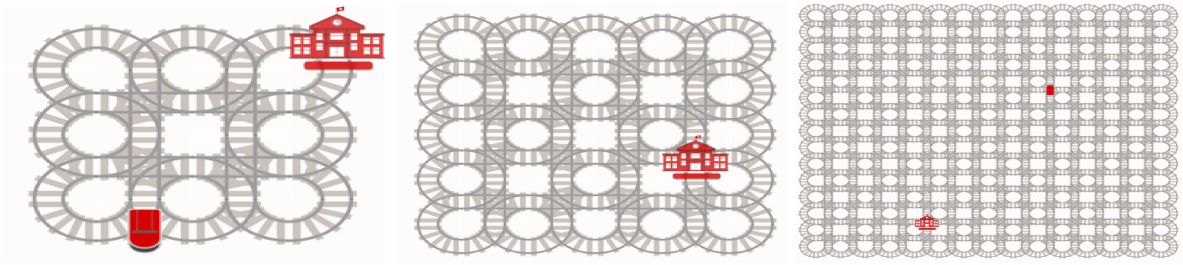


Figure 27: The Pathfinding Environment

Malfunction and Train-Speed-Difference Environment

Malfunctions and train speed differences are the same issue in their core when they are not causing any other problems. It is possible to teach the network on both problems in the same environment because they are solved the same way. Deciding if they drive behind each other, taking the risk of a malfunction/slow speed of the agent in front or taking a longer but less dependent on the flawless functioning of others.
To teach them these skills, we use the environment out of Figure 28, where all trains start on the same cell on the left, travel in the same direction to exclude deadlocks, and need to reach the same goal on the right. They can decide between staying behind each other or distributing the traffic on multiple tracks, which aren't the fastest.
Aside from that, it is important to teach the network how to deal with malfunctions and train speed differences, while in combination with other tasks, the malfunction rate and speed difference increase during each task and then drop to zero when a new one begins. Through this, we ensure that our agent can learn the task in its simplest form after handling malfunctions and different velocities.

Figure 28: The Malfunction/Train-Speed-Difference Environment

Deadlock Environment

As mentioned in 5.2.3 Deadlocks, the most appearing deadlock is the Common Deadlock. We can exclude the other two as a result of their infrequent and rare appearance. So the training environment will just represent the task of avoiding common deadlocks and nothing else.
To instruct our agents on how to avoid deadlocks, different implementations of the environment, displayed in Figure 29, are used. The starting points on the left are the endpoints for the agents on the right and vice versa.
Our agents must evade the other trains in this structure to reach their destination. We start with many switches and two agents, giving them multiple chances to avoid deadlocks, introducing more agents and fewer switches the further they are into the task.

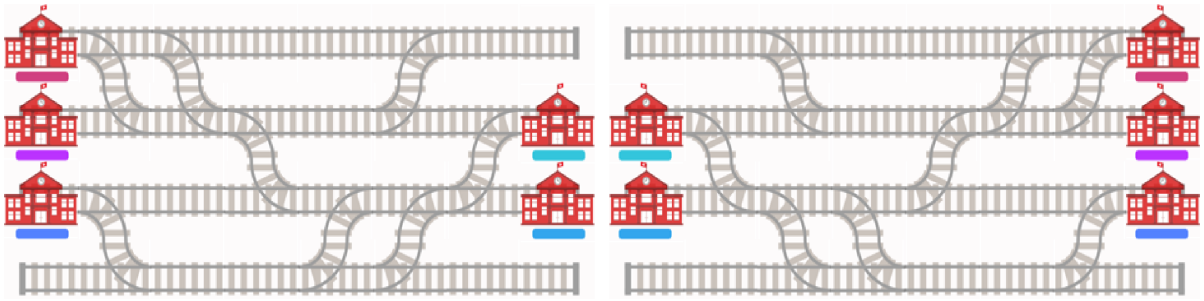


Figure 29: The Deadlock Environment

Curriculum Ordering

In the above, we specified the environments to teach our agents the central tasks in railway systems. But not only the environment but also the Sequencing(link zu CL) of these environments in which our network has to deal with them plays a crucial role.
We will compare two sequencing approaches, one without and the other with Rehearsal both shown in 5.3.3.
In the Custom Curriculum without Rehearsal, the different Tasks will appear after each other in increasing difficulty. On the other hand, older Tasks will reappear after a certain amount of network steps in the Custom Curriculum with Rehearsal.
We use this Division to examine the effects of rehearsal on the different algorithms and to compare their ability to retain earlier learned knowledge.
We decided to equally distribute the time a model spends in each environment.

Curricula Tables

Table 1: Environment composition for different curriculum strategies in RL. Each section corresponds to a specific curriculum type, detailing the environmental specifications, size, number of agents, train speed(4 possible speeds, with probability of each train having that speed in the column), malfunction characteristics, and the amount of network training steps.

No Curriculum					
Env	Size	Agents	TrainSpeed	Malfunction	Networksteps
sparse-generator	64x64	14	25/25/25/25	1/1000	1.000.000
Simple Curriculum					
Env	Size	Agents	TrainSpeed	Malfunction	Networksteps
sparse-generator	16x18	1	00/0/0/100	0	200.000
sparse-generator	28x28	3	50/00/0/50	1/1000	200.000
sparse-generator	40x40	6	50/00/25/25	1/1000	200.000
sparse-generator	52x52	10	25/25/25/25	1/1000	200.000
sparse-generator	64x64	14	25/25/25/25	1/1000	200.000
Custom Curriculum without Rehearsal					
Env	Size	Agents	TrainSpeed	Malfunction	Networksteps
Pathfinding	4x4	1	00/00/00/100	0	80.000
Pathfinding	8x8	1	00/00/00/100	0	80.000
Pathfinding	16x16	1	00/00/00/100	0	80.000
Pathfinding	32x32	1	00/00/00/100	0	80.000
Malfunction	12x5	5	00/00/00/100	1/100	80.000
Malfunction	12x5	6	50/00/00/50	1/100	80.000
Malfunction	12x5	7	50/00/25/25	1/100	80.000
Malfunction	12x5	8	25/25/25/25	1/100	80.000
Deadlock	32x2	2	00/00/00/100	1/64	80.000
Deadlock	32x2	4	50/00/00/50	1/64	80.000
Deadlock	32x4	8	50/00/25/25	1/64	80.000
Deadlock	32x4	16	25/25/25/25	1/64	120.000
Custom Curriculum with Rehearsal					
Env	Size	Agents	TrainSpeed	Malfunction	Networksteps
Pathfinding	4x4	1	00/00/00/100	0	65.000
Pathfinding	8x8	1	00/00/00/100	0	65.000
Pathfinding	16x16	1	00/00/00/100	0	65.000
Pathfinding	32x32	1	00/00/00/100	0	65.000
Malfunction	12x5	5	00/00/00/100	1/100	65.000
Malfunction	12x5	6	50/00/00/50	1/100	65.000
Malfunction	12x5	7	50/00/25/25	1/100	65.000
Malfunction	12x5	8	25/25/25/25	1/100	65.000
Pathfinding	32x32	1	00/00/00/100	0	65.000
Deadlock	32x2	2	00/00/00/100	1/64	65.000
Deadlock	32x2	4	50/00/00/50	1/64	65.000
Deadlock	32x4	8	50/00/25/25	1/64	65.000
Deadlock	32x4	16	25/25/25/25	1/64	65.000
Pathfinding	32x32	1	00/00/00/100	0	65.000
Malfunction	12x5	8	25/25/25/25	1/100	65.000
Deadlock	32x4	16	25/25/25/25	1/64	25.000

5.4 Adaptation to Non-Stationarity

5.4.1 Standard and Reset Upper Confidence Bound

An upper confidence bound (UCB)[64] is a statistical technique used in decision-making and optimization algorithms. We select an action dependent on a confidence value and the expected performance of an action.
The confidence of an action rises if an action is chosen more often and falls if an action is chosen less compared to other actions. UCB helps us balance the exploration of potentially better options that have not been tried that much with the exploitation of known good options, enabling our models to make informed decisions in uncertain environments that yield varying results. To employ the UCB exploration equation in an RL setting we utilize the following function for the Q-value:

$$Q_t(a) = \bar{Q}_t(a) + c\sqrt{\frac{\ln(t)}{N_t(a)}} \tag{21}$$

- $Q_t(a)$: Estimated value of action a at time t.
- $\bar{Q}_t(a)$: Average reward obtained from taking action a up to time t.
- $N_t(a)$: Number of times action a has been taken up to time t.
- c: Constant controlling the exploration-exploitation trade-off.
- t: Total number of time steps.

The standard UCB exploration assumes stationarity with actions that do not always deliver the same results due to an underlying probability distribution of outcomes, leading to possible problems in Non-Stationary environments. Especially actions that have been tried a lot in the start and did yield low rewards will likely not be explored for a longer time in the later environments as a result of the high confidence carried over from the start.
To counteract this weakness, we will adapt the standard UCB to address Non-Stationarity. Inspired by the RestartQ-UCB algorithm[65], which resets all its parameters in intervals and is, therefore, able to overfit well on these intervals, we adapted the standard UCB to reset its confidence bounds (setting $N_t(a)$ and t to 0) as soon as the environment changes in the Curriculum, to promote a faster understanding of the newly introduced task.

5.4.2 Elastic Weight Consolidation

Elastic Weight Consolidation (EWC) is a technique to adapt the loss function of our algorithms to make it harder to alter the weights of neurons in the neural network depending on their importance of performing in the current task. EWC enables our models to preserve the skill of acting reliably in past tasks after operating and learning new environments[66].

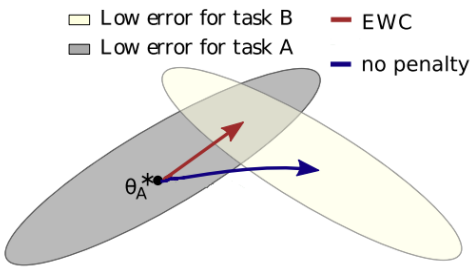


Figure 30: EWC ensures task A is remembered while training on task B[66]

By incorporating EWC, neural networks can display more robust learning across multiple tasks, reducing the risk of catastrophic forgetting and improving overall performance in continual learning scenarios, making EWC particularly useful when a model needs to attain new abilities without losing the knowledge acquired from previous experiences, displayed in Figure 30.

We implement EWC by adapting the standard loss function to the following equation:

$$L(\theta) = L_{current}(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{old,i})^2 \tag{22}$$

- $L(\theta)$: Total loss function
- $L_{\text{current}}(\theta)$: Current task's loss
- $\frac{\lambda}{2}$: Weight regularization term
- F_i: fisher information matrix : Importance of parameter i for the previous task
- $(\theta_i - \theta^*_{\text{old},i})^2$: Squared difference between the current and old parameter values for task i

The Fisher Matrix quantifies the importance of each weight by measuring how much changing a specific weight affects the overall loss function. It does this by computing the second-order derivatives of the loss function for each weight, reflecting the sensitivity of the model to changes in the individual weight.

5.4.3 Rational Padé Activation Unit

Adaptive Rational Activation Functions[67] also known as rational Padé Activation Units(RPAUs)are a type of learnable activation function used in neural networks. Unlike traditional activation functions that are based on simple mathematical operations like the sigmoid, hyperbolic tangent, or rectified linear unit function, Adaptive Rational Activation functions are designed based on rational functions.
A rational function is the quotient of two polynomials, making it able to capture more complex relationships between the input and output patterns of a neuron.
To make the rational function adaptive and therefore suited for Non-Stationarity, we can use the following equation[68]:

$$R(x) = \frac{\sum_{j=0}^{m} a_j x^j}{1 + \sum_{k=1}^{n} b_k x^k} \tag{23}$$

In this equation, a_j and b_k are parameters that are learned depending on the different inputs/rewards of the environment, making it a good fit for acting in Non-Stationary environments due to the ability of continuously adapting the activation function dependent on new incoming state-action-reward combinations. By changing n and m, we can define how complex the function can become.
We initialize the RPAUs to represent a rectified linear unit function, which is then adapted during training, this process of the changing activation function is displayed in Figure 31.

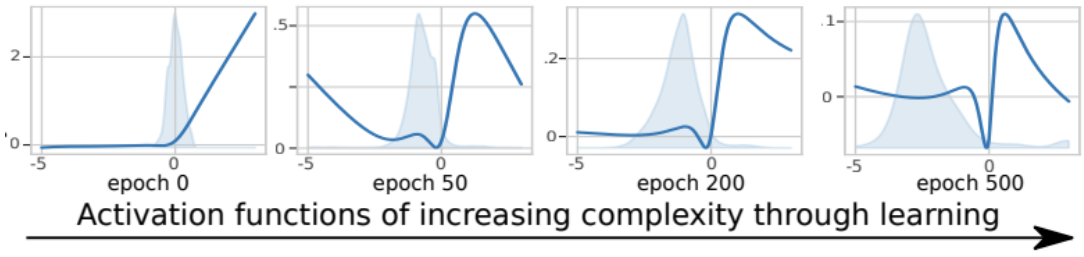


Figure 31: The Adaptive Rational Activation Functions changing dependent on the input distribution and getting complexer over time[67]

6 Experimental Analysis

6.1 Experimental Setup

6.1.1 Evaluation Environments

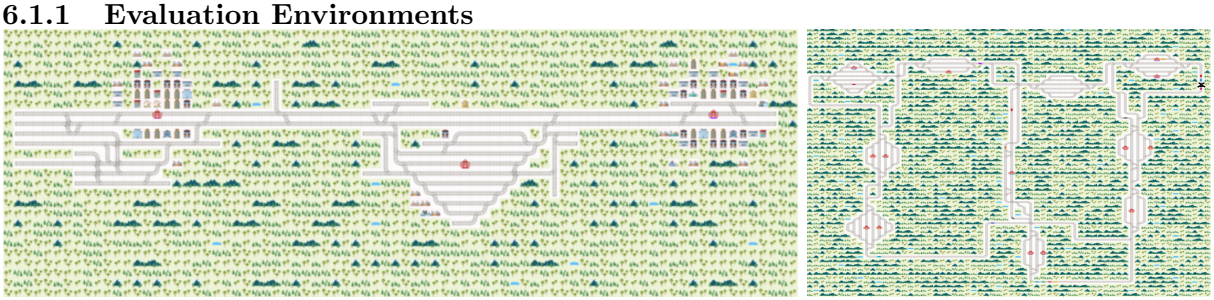


Figure 32: Possible implementations of the environment

To assess the baseline performance and their adaptations when combined with curricula, we will employ two distinct environments. The initial environment is a 64x64 evaluation setting with 14 agents illustrated in Figure 32 on the right, being the same environment as in the No Curriculum and the final Simple Curriculum environment.
The second environment, as depicted in Figure 32 on the left, is a custom evaluation environment with 8 agents, that mimics a more realistic railway structure[69]. In this environment, our algorithms will be evaluated without prior training exposure, highlighting their capability to generalize and apply learned skills in unknown environments.

6.1.2 Metrics and Plots

We will use score and done rate as metrics to evaluate the experiments. The score is the combined reward all agents receive at the end of an episode divided by the number of agents to ensure it still lies between zero and minus one. This adjustment is for comparable results in different environments where agent counts may vary. The done rate is the percentage of trains reaching their destination before the simulation ends.
Both of these metrics will be plotted to network steps, which represent the amount of network updates a model takes to adapt its behavior. We chose to use network steps because the different environments can have massively different sizes and agent counts, leading to fluctuating knowledge gain per environment.
It is necessary to mention that a higher score does not always correlate with a higher done rate. This is because the score represents how close an agent got to its destination. In most cases, we need rather large improvements in the done rate to get relatively small gains in the score, especially at the higher end.
Each training was repeated three times, with each iteration comprising 50 runs on the designated evaluation environment. Consequently, a total of 150 data samples were generated for each algorithm curriculum combination.
The combination of this amount of data samples, coupled with the deadlock cascading characteristic of the TSP and the relatively low agent count in the evaluation environments (8 and 14, respectively), fosters significant variation in both the score and, particularly, the done rate.
For example, if one out of eight trains is misrouted in the custom evaluation environment, leading to a deadlock, the done rate drops by at least 0.25 when only the misrouted train and another are affected.
In the plots, a straight vertical line indicates the change from one task to another, for example, from the pathfinding environment to the deadlock environment.
On the other hand, a vertical dotted line represents the change of an environment but staying in the same task, for example, when changing from the 4x4 pathfinding environment to the 8x8 pathfinding environment.
All results of the evaluations are recorded in Table 5.3.3

6.1.3 Model Parameters

Our baselines consist of an Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO), and Deep-Double-Dueling-Q algorithm(DDDQN) algorithm, which utilize two hidden layers with a size of 512 each, a replay buffer of the size 1000000, a batch size of 128 and a learning rate of 0.00005.
When not stated that we are using another exploration strategy, we will use a simulated annealing exploration strategy with the following formula giving us the probability of choosing an exploring action:

$$a_{\text{explore}} = \frac{1}{(1.000005)^{\text{networksteps}}} \tag{24}$$

With our three adaptations to Non-Stationarity, we will use:

- the value 1 for c, the Constant controlling the exploration-exploitation trade-off in the upper confidence bound(UCB) exploration
- the value 2 for λ, setting how much the elastic weight consolidation(EWC) can hinder the further adaption of certain neurons
- the value 5 for m and 4 for n, for the Rational Padé Activation Unit(RPAU), which we initialize with the shape of a Rectified Linear Unit.

6.1.4 Experiments

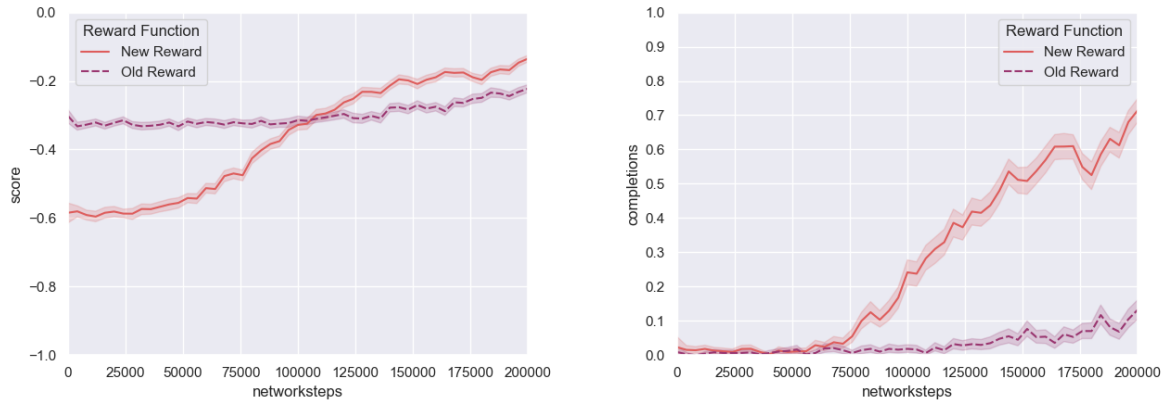
In Section 6.2, we first examine the effects of changing the reward function to remove the unwanted complications introduced by a sparse reward setting. The standard reward function just gave our agents rewards based on the end state. The closer an agent was to its destination, the higher the reward it received, with zero being the highest score and minus one being the lowest score each agent could achieve.
With pathfinding being the most important skill on which all others build, we decided to implement a reward at every timestep dependent on the relative direction an agent is traveling. If an agent is closer to his target at the present timestep compared to his last timestep, he receives a reward of zero and a reward of minus 0.05 otherwise. The reward based on the end state stays the same, leaving us with the following equation for our agents, which we will use for all other experiments:

$$r_a = \begin{cases} r_a = a_{\text{shortest path}}(t) & \text{if } t = T \\ r_a = 0 & \text{if progress}(t) > 0 \\ r_a = -0.05 & \text{else} \end{cases} \tag{25}$$

$$\text{progress}(t) = \text{shortest path}(t-1) - \text{shortest path}(t) \tag{26}$$

In Section 6.3, we will observe the training processes of our baselines in different curricula, meaning comparing DDDQN, PPO, and A2C with each other in the No, Simple, Custom, and Custom with Rehearsal Curriculum.
In Section 6.4, we will evaluate the performances on the 64x64 and the Custom environment, to research the effects of the curriculum on the baselines and to compare the evaluation results between the two environments.
In Section 6.5, we will adjust the most successful algorithm of the baselines with the Non-Stationarity adaptations from Section 5.4 to test if these are able to leverage the positive effects of the curriculum.

6.2 Reward Function



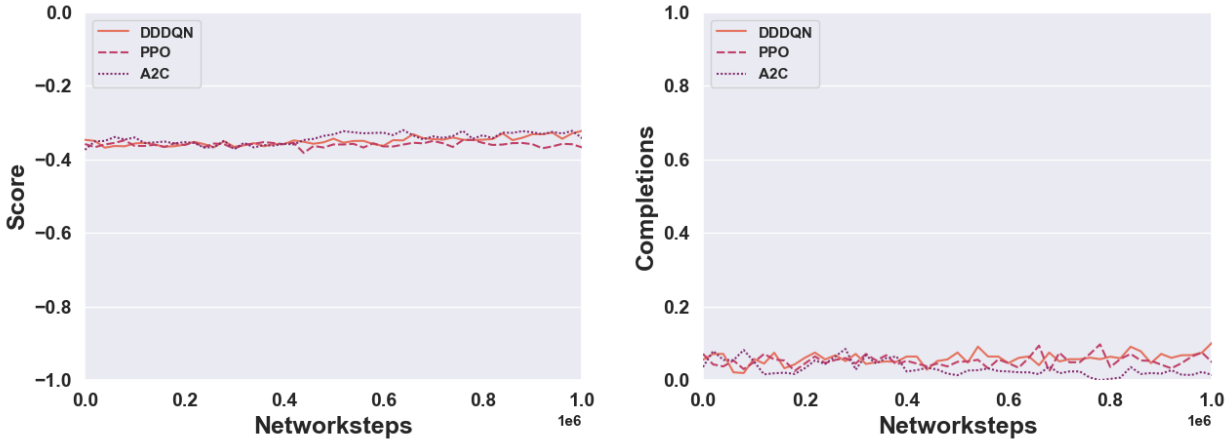
(a) Score (b) Done-Rate

Figure 33: Comparison of the new reward function(orange continuous line) with the old reward function(dashed purple line) performance during training of the DDDQN algorithm in the pathfinding environment

We compare the standard with the adapted new reward functions in the pathfinding environment of the size 32x32 with one agent on their score and their done-rate in Figure 33 with the basic DDDQN algorithm.
We can observe that the done rate of the new reward function is much more correlated with its score than the old function. In addition to that, when using the adapted reward function, the model achieves a done rate of multiple factors higher than the old reward function given the same time to train, making it much more efficient.
For the following experiments, we will use the adapted reward function.

6.3 Relative Performance of the Algorithms in different Curricula

6.3.1 No Curriculum

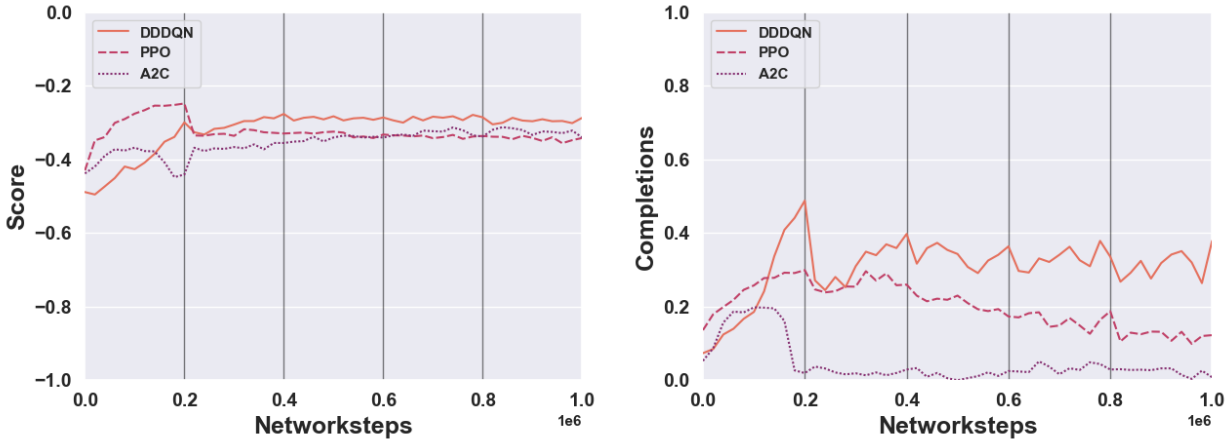


(a) Score (b) Done-Rate

Figure 34: Performance comparison in score and done rate during training with No Curriculum employed between DDDQN(continuous orange line), PPO(dashed red line), and A2C(dotted purple line).

While training with No Curriculum over the course of one million network steps, the three baseline algorithms demonstrated no to little improvements in their score and done-rate.
Notably, the DDDQN and A2C outperformed the PPO by improving slightly over time with its score in Figure 34.
When comparing the done rate in Figure 34, we can observe low completions across all algorithms, meaning that nearly no agents reach their destination. Despite the ability of the A2C to outperform PPO in the score, its done rate is lower than that of the other two, leading us to the assumption that A2C was able to exploit the reward function to receive high rewards without actually bringing the agents to their goal.
The results cast doubt on the effectiveness of training RL agents without a curriculum in the given environment. The lack of a structured learning path appears to hinder the network's ability to learn, as seen in the lack of improvement over time, especially in the completions.

6.3.2 Simple Curriculum



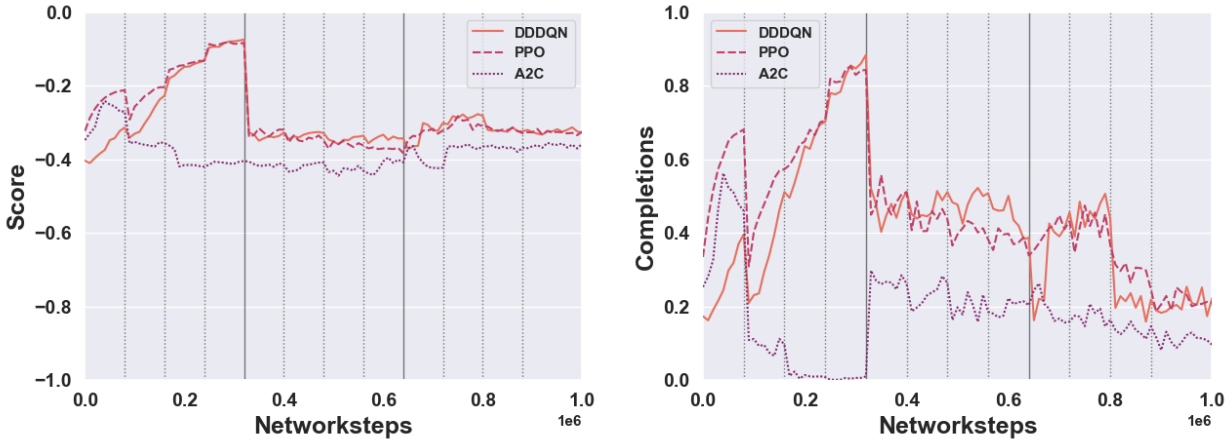
(a) Score

(b) Done-Rate

Figure 35: Performance comparison in score and done rate during training with the Simple Curriculum employed between DDDQN(continuous orange line), PPO(dashed red line), and A2C(dotted purple line).

When deploying the Simple Curriculum, we can observe different scores in Figure 35 right from the start in the first environment. While DDDQN and PPO improve their score relatively fast over the first 200.000 network steps, A2C struggles to improve, with its done rate and score plummeting near the end of the first environments.
In the second environment, all algorithms have a noticeable drop in performance, probably caused by the increased complexity introduced by raising the agent count from one to three. From the second environment onward, DDDQN outperforms the other two algorithms by holding roughly the same performance in increasingly complex environments while the performance of the other two algorithms slowly declines.
The done rates in Figure 35 reveal the varied abilities of the algorithms to reach their goals in the Simple Curriculum. Overall timesteps, DDDQN acquires the highest done rate, while A2C performs the worst, and PPO achieves done rates between the two.
Comparable to the performances without a curriculum, A2C was able to exploit the reward function, achieving similar scores to PPO while having the lowest done rate.

6.3.3 Custom Curriculum without Rehearsal

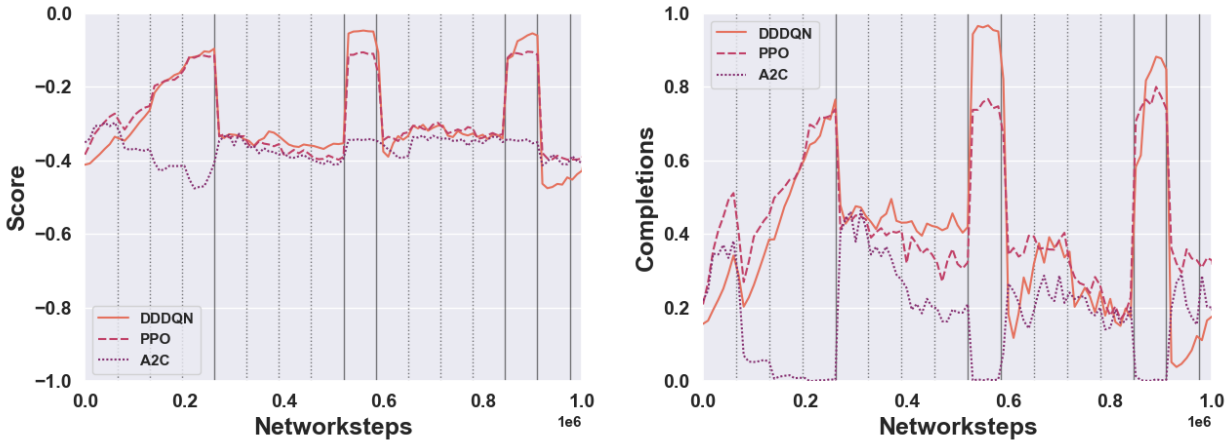


(a) Score (b) Done-Rate

Figure 36: Performance comparison in score and done rate during training with the Custom Curriculum without Rehearsal employed between DDDQN(continuous orange line), PPO(dashed red line), and A2C(dotted purple line).

The training results of the Custom Curriculum without Rehearsal in Figure 36 show that it is rather easy to get high scores in the first pathfinding environment. While the A2C can not keep up with more complex pathfinding tasks after the first one, as seen in the decreasing score and massively decreasing done rate, PPO and DDDQN are slightly improving their score and done rate after a big decrease after the first increase in difficulty.
In the malfunction environment, PPO and DDDQN have roughly the same score and the done rate, with DDDQN slightly outperforming PPO. A2C performs much worse than the other two algorithms in both metrics. Overall, there is a lot of fluctuation due to the increase in complexity.
In the deadlock environment, the done rate and score decline after the third deadlock environment. A2C performance steadily decreased while the performance of PPO and DDDQN stabilized at roughly the same level after the third environment.

6.3.4 Custom Curriculum with Rehearsal

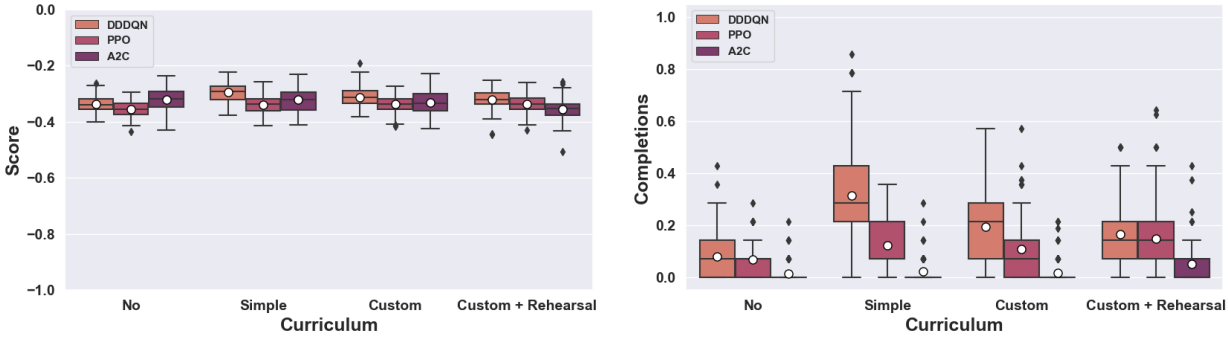


(a) Score (b) Done-Rate

Figure 37: Performance comparison in score and done rate during training with the Custom Curriculum with Rehearsal employed between DDDQN(continuous orange line), PPO(dashed red line), and A2C(dotted purple line).

The performance of the baselines in the Custom Curriculum with Rehearsal(Figure 37) is analogous to the Custom Curriculum without Rehearsal for the first 520.000 network steps due to just some minor differences in network steps in each environment.
After step 520.000, the pathfinding environment is rehearsed for the first time. DDDQN is able to achieve a score and done rate higher than the last time it was acting in the same environment. PPO performs roughly the same ways as before, and the A2C algorithm is still not able to have a completion rate above zero in the most complex pathfinding environment.
In the deadlock environment, the algorithms act differently at the start, but after the initial differences, they all approach roughly the same score and rate with increasing environment complexity.
The second rehearsal of the pathfinding environment is similar to the first rehearsal for PPO and A2C. DDDQN exhibits a slightly worse performance in the done rate but no decrease in the score.
In the rehearsal of the malfunction environment, PPO and A2C outperform DDDQN in both metrics, while PPO is able to reach a higher done rate with a similar score as A2C.
In the rehearsal of the deadlock environment, the baselines exhibit the same behavior as in the rehearsal of the malfunction environment.

6.4 Comparison in Performance of the Baselines

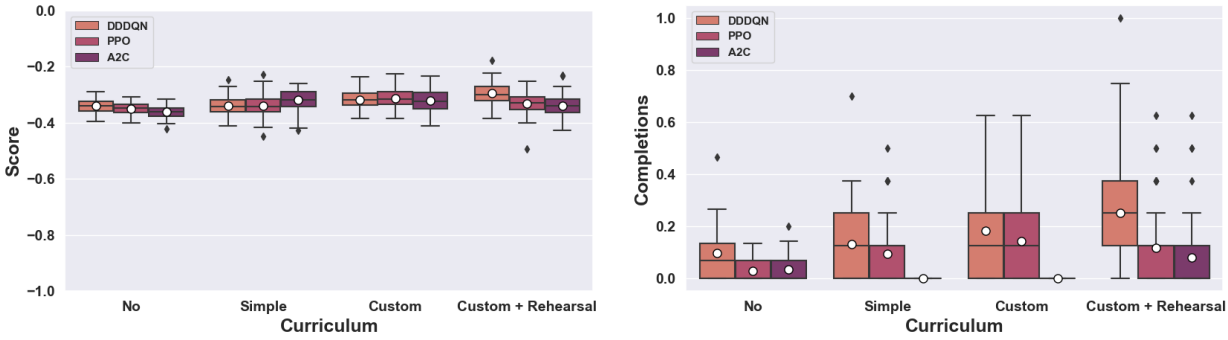


(a) Score

(b) Done-Rate

Figure 38: Comparison of performance in score and done rate of DDDQN(orange), PPO(red), and A2C(purple) evaluated in the 64x64 environment grouped by the used curricula.

In Figure 38, you can see the scores and completion rate of the baseline algorithms with the respective curriculum evaluated on the 64x64 evaluation environment.
We can observe that the scores roughly correlate with the done rate over all curricula, except the No Curriculum, where the A2C reached the highest mean score among the baselines while having a mean done rate of near zero, which further supports the assumption that the A2C is better at exploiting the reward function receiving high scores without reaching destinations.
In the done rate, we can see clear differences between the curricula and the baselines within each curriculum. DDDQN outperforms PPO, and PPO surpasses A2C in the mean completion rate in all Curricula. The data showcases a clear trend indicating the superior effectiveness of DDDQN compared to both PPO and A2C in the TSP.
The Simple Curriculum paired with the DDDQN algorithm achieves the highest mean completion rate. This can be explained by the curriculum introducing the evaluation environment in a feasible manner by building on simpler versions of the evaluation environment.
Both the Custom Curricula with and without Rehearsal exceed the No Curriculum in the done rate even though the Custom Curricula have not seen this environment in their training, implying a generalization capability.



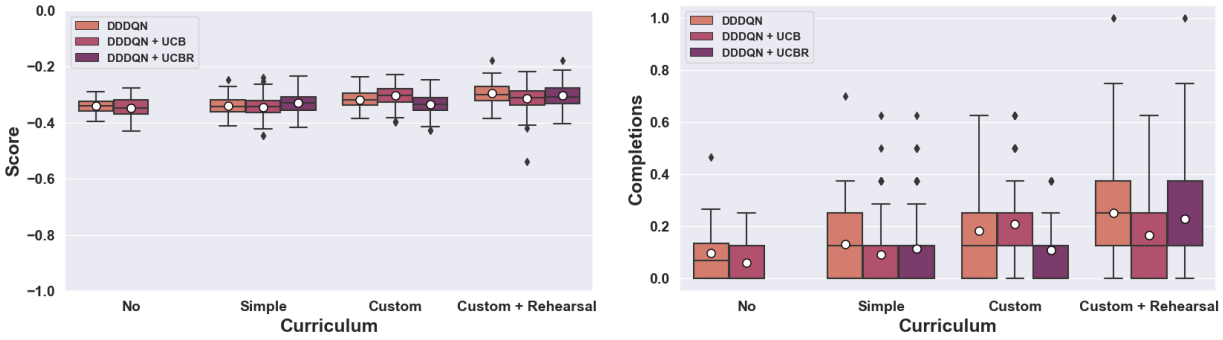
(a) Score (b) Done-Rate

Figure 39: Comparison of performance in score and done rate of DDDQN(orange), PPO(red), and A2C(purple) evaluated in the custom environment grouped by the used curricula.

In Figure 39, you can see the scores and completion rate of the baseline algorithms with the respective curriculum evaluated on the custom environment. This environment has not been seen in training by any of the evaluated algorithms.
Like in the 64x64 environment evaluation, DDDQN is also performing with a higher mean done rate than the PPO and A2C in this environment in every curriculum. The main difference in the evaluation of the 64x64 environment can be seen when comparing the different Curricula with each other. Other than before, both custom curricula achieve higher mean done rates than the Simple Curriculum, supporting the argument that the custom curricula have better generalization capabilities than the No/Simple Curriculum. The performance of the Custom Curriculum is noticeably improved with rehearsal implemented in the training, indicating that the model partially forgets skills learned in earlier environments and that rehearsal prevents catastrophic forgetting to a certain extent.
With DDDQN surpassing the other two baselines in all mean done rate evaluations and marginally exceeding the others in most of the score evaluations.
While proceeding with the adaptation of DDDQN to Non-Stationarity, we will exclude PPO and A2C. This decision is driven by the in average higher means in the results, recognizing that the outcomes have high variance.

6.5 DDDQN adapted to the Non-Stationarity

6.5.1 Upper Confidence Bound

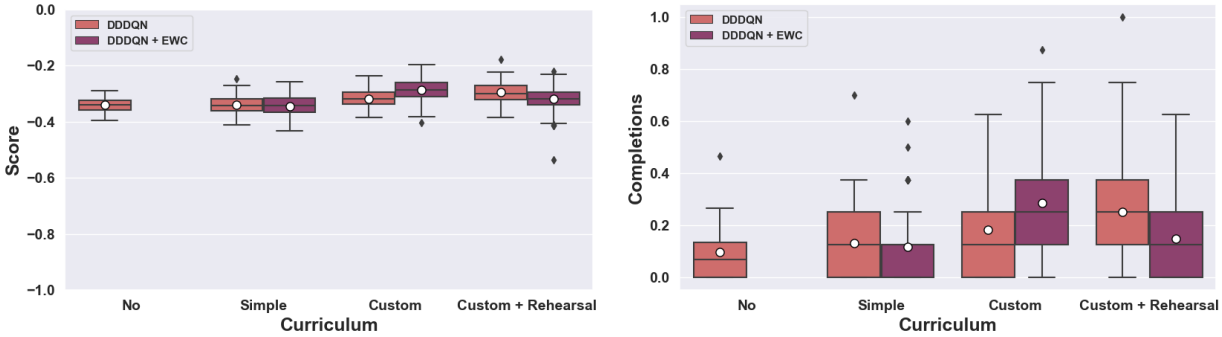


(a) Score (b) Done-Rate

Figure 40: Comparison of performance in score and done rate of the standard DDDQN(orange), DDDQN with UCB exploration(red), and DDDQN with reset UCB exploration(purple) evaluated in the custom environment grouped by the used curricula.

In Figure 40, we can identify that both UCB exploration strategies decrease the mean done rate of the DDDQN algorithm in all curricula except the standard UCB exploration combined with the Custom Curriculum, which brings a minor increase in the done rate.
These results are attributable to the fact that after the pathfinding environment, exploration is implicit when choosing an exploiting action. This is the case because, in the pathfinding environment, the model learned that the action with the highest return is to take the shortest path to its destination. When following this behavior in the subsequent environments, the reward of the action choosing the shortest path will be less than alternative actions due to ending up in deadlocks or behind malfunctioning trains, leading our model to choose other actions which still have higher Q-values.

6.5.2 Elastic Weight Consolidation



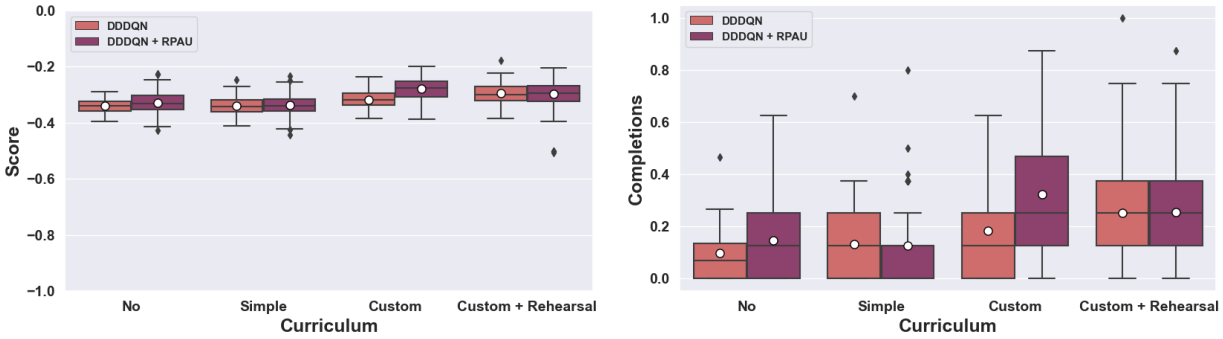
(a) Score

(b) Done-Rate

Figure 41: Comparison of performance in score and done rate of the standard DDDQN(orange) and DDDQN with elastic weight consolidation(purple) evaluated in the custom environment grouped by the used curricula.

The standard DDDQN algorithm is compared with DDDQN utilizing EWC in Figure 41. EWC can not be sensibly deployed in the No Curriculum due to the absence of changing tasks/environments and the resulting problem of deciding how to divide a non-changing environment, and therefore, we did not evaluate DDDQN with EWC on the No Curriculum.
The algorithm does not display a massively changed behavior in the Simple Curriculum when deploying EWC, probably due to the parameters that are locked by the EWC being of roughly the same importance in all environments.
The DDDQN with EWC exhibits an improved mean score and mean done rate in the Custom Curriculum without Rehearsal. This improvement can be attributed to the EWC locking the weights after each environment, reducing catastrophic forgetting, and enabling our model to retain the information of each environment and utilize these multiple skills jointly in the evaluation environment.
In the Custom Curriculum with Rehearsal, we observe a significant decrease in both mean score and mean completion rate when we include EWC. We can attribute this phenomenon to the reintroduction of older environments towards the end of the subtask. Since the model engages with a different environment before consolidating weights, it does not solely reflect the crucial aspects of the last task; instead, it inefficiently combines information from two to three tasks and locks the weights/biases based on this information.

6.5.3 Rational Padé Activation Units

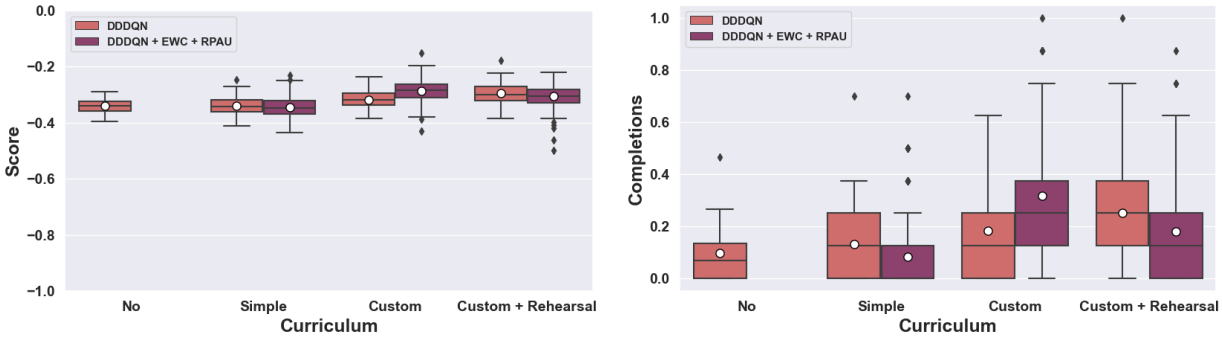


(a) Score (b) Done-Rate

Figure 42: Comparison of performance in score and done rate of the standard DDDQN(orange) and DDDQN with RPAUs, evaluated in the custom environment grouped by the used curricula.

The evaluation results of the DDDQN with RPAUs compared to the standard DDDQN in Figure 42 show us that the RPAUs do not lead to a decrease in mean score or mean done rate.
In the No Curriculum, we have improved mean done rate and mean score because the algorithm can map more complex relations between the input and the output due to the increased complexity in the activation function of the neurons.
There is no change in mean score or mean done rate in the Simple Curriculum/Custom Curriculum with Rehearsal worth mentioning.
In the Custom Curriculum without Rehearsal, we can see an improvement in the mean done rate, possibly ascribable to the increased complexity introduced by the adaptive rational activation function in combination with the structured learning environment. This combination enables the adaptive rational activation function to adapt the function customized to represent the different tasks.

6.5.4 Elastic Weight Consolidation with Rational Padé Activation Units



(a) Score (b) Done-Rate

Figure 43: Comparison of performance in score and done rate of DDDQN(orange) and DDDQN with RPAUs and EWC(purple), evaluated in the custom environment grouped by the used curricula.

With the results of EWC and RPAU combined in Figure 43, we can not see more improvement compared to deploying these techniques alone. The situation likely arises from the simultaneous enhancement of the network by the EWC and the adaptive rational activations functions, leading to conflicting effects.
The weights locked by EWC are selected based on the current activation function, which is subsequently altered by the adaptive activation function, making the weight consolidation invalid to a certain degree. Exploring specific parameter combinations may provide better results.

Tables

Table 2: Evaluation Results showcasing the performance of various algorithms in combination with multiple curricula, split by the evaluation environments. The table includes mean scores, standard deviations, completion metrics, and rate of improvement over the baseline algorithm completions in the No Curriculum.

64x64 Environment Evaluation		Scores	Completions	
Curriculum	Algorithm	mean±*std*	mean±*std*	Impr
No	DDDQN	-0.337 ±0.027	0.079 ±0.093	0%
	PPO	-0.355 ±0.028	0.067 ±0.064	0%
	A2C	-0.322 ±0.039	0.012 ±0.036	0%
Simple	DDDQN	-0.294 ±0.035	0.312 ±0.200	295%
	PPO	-0.338 ±0.030	0.122 ±0.094	82%
	A2C	-0.322 ±0.041	0.021 ±0.048	75%
Custom	DDDQN	-0.314 ±0.032	0.192 ±0.132	143%
	PPO	-0.337 ±0.029	0.116 ±0.113	73%
	A2C	-0.332 ±0.041	0.016 ±0.045	33%
Custom + Rehearsal	DDDQN	-0.321 ±0.033	0.165 ±0.114	101%
	PPO	-0.336 ±0.032	0.148 ±0.128	120%
	A2C	-0.355 ±0.034	0.052 ±0.077	333%
Custom Environment Evaluation		Scores	Completions	
Curriculum	Algorithm	Mean ±*std*	Mean±*std*	Impr
No	A2C	-0.362 ±0.024	0.033 ±0.051	0%
	PPO	-0.353 ±0.021	0.029 ±0.042	0%
	DDDQN	-0.339 ±0.027	0.097 ±0.099	0%
	DDDQN+UCB	-0.346 ±0.034	0.061 ±0.089	-37%
	DDDQN+RPAU	-0.328 ±0.036	0.144 ±0.138	48%
Simple	A2C	-0.318 ±0.038	0.000 ±0.000	-%
	PPO	-0.339 ±0.036	0.093 ±0.113	221%
	DDDQN	-0.338 ±0.032	0.131 ±0.129	35%
	DDDQN+UCB	-0.343 ±0.034	0.091 ±0.117	-6%
	DDDQN+UCBR	-0.332 ±0.035	0.115 ±0.134	19%
	DDDQN+EWC	-0.343 ±0.039	0.116 ±0.114	19%
	DDDQN+RPAU	-0.336 ±0.035	0.126 ±0.123	29%
	DDDQN+RPAU+EWC	-0.344 ±0.038	0.081 ±0.121	-7%
Custom	A2C	-0.321 ±0.037	0.000 ±0.000	-%
	PPO	-0.313 ±0.034	0.142 ±0.141	340%
	DDDQN	-0.317 ±0.033	0.184 ±0.157	90%
	DDDQN+UCB	-0.303 ±0.036	0.208 ±0.164	114%
	DDDQN+UCBR	-0.333 ±0.034	0.107 ±0.112	10%
	DDDQN+EWC	-0.287 ±0.036	0.286 ±0.195	195%
	DDDQN+RPAU	-0.279 ±0.038	0.322 ±0.209	232%
	DDDQN+RPAU+EWC	-0.285 ±0.039	0.317 ±0.212	227%
Custom + Rehearsal	A2C	-0.342 ±0.036	0.081 ±0.124	145%
	PPO	-0.331 ±0.037	0.117 ±0.143	304%
	DDDQN	-0.294 ±0.036	0.252 ±0.186	160%
	DDDQN+UCB	-0.314 ±0.041	0.166 ±0.148	71%
	DDDQN+UCBR	-0.302 ±0.039	0.228 ±0.192	135%
	DDDQN+EWC	-0.318 ±0.038	0.149 ±0.146	53%
	DDDQN+RPAU	-0.298 ±0.044	0.253 ±0.191	160%
	DDDQN+RPAU+EWC	-0.305 ±0.04	0.181 ±0.173	86%

7 Discussion

7.1 Conclusion

In this thesis, we deconstructed the TSP into its subproblems and designed environments from the most important ones. These environments were then used to build a Custom Curriculum and an adapted version, which also employed rehearsal of the different environments.
When evaluating on the 64x64 evaluating environment, both custom curricula surpass the No Curriculum in mean done rate but could not keep up with the Simple Curriculum, which is designed to perform on the 64x64 evaluation environment and has seen it during training.
When evaluating the models in the custom environment, which has not been seen in any of the curricula, using No Curriculum performs the worst, and the Simple Curriculum gets outdone by both the custom curricula. This mean indicates, on the one hand, that our models were able to better generalize with the Custom Curriculum to unseen environments and, on the other hand, that the models trained with the Simple Curriculum were not or only partially learning the actual skill of train scheduling and instead overfitted to the environment giving by the sparse_generator. Additionally, when utilizing rehearsal, the custom curricula demonstrated increased mean completions, suggesting that rehearsal is a strong tool to counteract catastrophic forgetting in artificial neural networks.
After the evaluation and comparison of the baseline in different curricula, we proceeded to adapt DDDQN to the Non-Stationarity adaptations, which indeed leveraged the effects by improving the mean done rate of the curricula in certain cases.
The UCB and the RestartUCB both were not able to noticeably improve our results, probably due to the redundancy of the UCB exploration strategy, as a cause of the implicit exploration of exploiting actions created with the buildup of the curricula.
The EWC improved the mean results of the DDDQN when training with the Custom Curriculum without Rehearsal by 195%. We could not see equal results in the Custom Curriculum with Rehearsal, explainable by the rehearsal befouling the importance of each neuron to the task.
The most improvement of the mean done rate was achieved when deploying the RPAUs in combination with the Custom Curriculum, leading to a 232% increase in mean completions when compared to the standard algorithm without a curriculum. This can be attributed to the greater capability of RPAUs in mapping input-output pairs, enabled by the increased complexity of their activation function. Further, this complexity evolves during training, rendering it responsive to the curriculum and enabling the curriculum to guide the model in efficiently establishing input-output connections.
This thesis showed promising results regarding the power of utilizing Non-Stationarity adaptations in CL, making the models able to generalize better and learn a skill much better in the same amount of training time, however, additional investigation is needed, given the high variance across almost all results, prompting a degree of skepticism.

7.2 Future Work

For future work and the space of ML, it is crucial to go beyond assessing a model's performance post-training. We should also focus on enhancing the model's learning efficiency, exploring methods to train it effectively and achieve high performance while having less training time and/or utilizing smaller ANNs. This is similar to the optimization efforts in OR, where we aim to decrease runtime complexity while maintaining the precision of algorithms.
And with the current interest in RL and AI as a whole, we should not lose OR or combinations between OR and RL out of sight as possible solutions in the future.
As the last point, it is important to validate if agents are robustly learning a relevant skill or if they are memorizing certain trajectories of their training environment and are overfitting to this specific training domain, not being able to grasp the core of the task. This can be done by evaluating the model on environments that require the same skills as needed in the training but are not constructed the same way.

7.3 Limitations

7.3.1 High Variance in Results

One of the main limitations of this thesis is the high variance across all results. Due to this being a critical point to discuss, we will address this topic and explain how high variance is promoted in the experiments, partially diminishing the strength of the claim that the results are rendered invalid by the high variance.
First, train scheduling is very vulnerable to errors of a single train, causing one or more trains to not reach their destination, promoting higher variance. Should our algorithms make a less-than-optimal decision, it could set off a chain of consequences, affecting other trains and their arrival capabilities.
Second, we also increase variance with malfunctions, introducing a probability of hindering a train from traveling further. Even if our algorithms learn to perfectly deal with malfunctions, the train malfunctioning does not arrive at its target in most cases.
Third, the low agent count paired with our metrics leverages the first two effects. With eight agents in our custom evaluation, the done rate can be one of nine discrete values(0,0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1), leading to one simple deadlock involving just two trains reducing the done rate by 0.25, making the results fluctuate a lot.
Last, we could have done an increased amount of training runs with more evaluation runs to reduce variance by having more data samples.

7.3.2 The Simulator

Even though the Flatland Simulator is a sufficient representation of the TSP, it is still a significant simplification of the real-life TSP. There are a lot of problems and solutions that aren't considered or heavily reduced in complexity for this simulator.
First, Malfunctions are a simple concept to model a combination of different error sources like power outages, medical emergencies, engine failures, and many more, which all depend on parameters like weather, age of the train, and location.
Second, partial observability is not considered at all. Trains aren't always able to detect their whole environment, leading to increased complexity of the task. Environment factors from rain or fog that limit vision or missing communication between trains, where one train is not able to receive crucial data from the other trains, needing the model to act with less data available than usual.
Last, the environment after initialization is static. The trains have fixed speeds with instant acceleration and no braking distance. We can't use resting trains to take over the assignments of others that are delayed or canceled, which is common practice in the real-life scheduling of passenger trains.
Overall, an efficient simulator that is capable of representing the real-world TSP as closely as possible would benefit this problem immensely. First, it could serve as a benchmark, making different approaches comparable, and second, it could reduce the gap between the simulation and real-world deployment.

7.3.3 The Curricula

The Curricula in use were limited. Even though we explored CL and CL with Rehearsal, we did not research how other factors influence the model's behavior and how important the single environments were for the end performance when evaluating. Adaptations that can be made to the different curricula are equally interesting as other adaptations but were not in the scope of this thesis.
These adaptations include the duration a model spends in each environment, the sequencing of the environments, and the environments that were in use.

Additionally, we could have explored dynamic Curricula, where we change the environments based on the performance of the model, like teacher-student CL, or similar to some extent, reintroduce older environments to rehearse if the model decreases in performance.

7.3.4 The Algorithms

The algorithms we used in this thesis were standard, well-researched algorithms with fixed network parameters and some adapted forms of the DDDQN to address the Non-Stationarity. We did not investigate the interaction between the number of neurons and/or layers with EWC and RPAU, where possibly further improvements can be made. In addition, we just tried a single parameter for each adaptation, missing out on refinements to our models.
But with RL and CL both being similar to the human learning process, it is close to hand to consider growing ANNs. These ANNs grow and adapt conditionally to their surroundings and on their advancement in different environments. By utilizing growing networks, we could even further leverage the effects of CL.
Another possible technique to improve generalization and adaptability is to prune the different neurons of the ANN. Research has shown that this practice reduces the risk of overfitting.

References

1. *EU Rail Projects for 2026* Accessed: 2024-01-19. https://rail-research.europa.eu/eu-rail-projects/.
2. Bansal, V., Kumar, V., Chopra, V., Jain, V. & Saxena, R. Effective Railway Planning Through operations Research. *International Journal of Innovative Science and Research Technology* **2** (2017).
3. Yan, Y. *et al.* Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities. *Transportation Research Part E: Logistics and Transportation Review* **162,** 102712 (2022).
4. Khetarpal, K., Riemer, M., Rish, I. & Precup, D. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research* **75,** 1401–1476 (2022).
5. Zhang, C., Vinyals, O., Munos, R. & Bengio, S. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893* (2018).
6. Papoudakis, G., Christianos, F., Rahman, A. & Albrecht, S. V. *Dealing with Non-Stationarity in Multi-Agent Deep Reinforcement Learning* 2019. arXiv: 1906.04737 [cs.LG].
7. Wang, X., Chen, Y. & Zhu, W. A survey on curriculum learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **44,** 4555–4576 (2021).
8. Mohanty, S. *et al.* Flatland-rl: Multi-agent reinforcement learning on trains. *arXiv preprint arXiv:2012.05893* (2020).
9. Wälter, J. Existing and Novel Approaches to the Vehicle Rescheduling Problem (VRSP): In the Course of the Flatland Challenge by Swiss Federal Railways (SBB). *University of Applied Sciences Rapperswil, Rapperswil, Switzerland,* 1–68 (2020).
10. Tesauro, G. *et al.* Temporal difference learning and TD-Gammon. *Communications of the ACM* **38,** 58–68 (1995).
11. Wang, W. *et al. From few to more: Large-scale dynamic multiagent curriculum learning* in *Proceedings of the AAAI Conference on Artificial Intelligence* **34** (2020), 7293–7300.
12. Lotter, W., Sorensen, G. & Cox, D. *A multi-scale CNN and curriculum learning strategy for mammogram classification* in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: Third International Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, Held in Conjunction with MICCAI 2017, Québec City, QC, Canada, September 14, Proceedings 3* (2017), 169–177.
13. Li, Y. Reinforcement learning in practice: Opportunities and challenges. *arXiv preprint arXiv:2202.11296* (2022).
14. Puterman, M. L. Markov decision processes. *Handbooks in operations research and management science* **2,** 331–434 (1990).
15. Bertsekas, D. *Reinforcement learning and optimal control* (Athena Scientific, 2019).
16. Kaiser, L. *et al.* Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374* (2019).
17. *OpenAISpinningUP* https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html. Accessed: 2024-01-18.
18. Prudencio, R. F., Maximo, M. R. & Colombini, E. L. A survey on offline reinforcement learning: Taxonomy, review, and open problems. *IEEE Transactions on Neural Networks and Learning Systems* (2023).
19. Sutton, R. S. & Barto, A. G. *Reinforcement learning: An introduction* (MIT press, 2018).

20. François, O. Global optimization with exploration/selection algorithms and simulated annealing. *The Annals of Applied Probability* **12,** 248–271 (2002).

21. Busoniu, L., Babuska, R. & De Schutter, B. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **38,** 156–172 (2008).

22. Mnih, V. *et al.* Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

23. Wang, Z. *et al. Dueling network architectures for deep reinforcement learning* in *International conference on machine learning* (2016), 1995–2003.

24. Van Hasselt, H., Guez, A. & Silver, D. *Deep reinforcement learning with double q-learning* in *Proceedings of the AAAI conference on artificial intelligence* **30** (2016).

25. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

26. Grondman, I., Busoniu, L., Lopes, G. A. & Babuska, R. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **42,** 1291–1307 (2012).

27. Simões, D., Lau, N. & Reis, L. P. Multi-agent actor centralized-critic with communication. *Neurocomputing* **390,** 40–56 (2020).

28. Foerster, J. N., Assael, Y. M., de Freitas, N. & Whiteson, S. *Learning to Communicate to Solve Riddles with Deep Distributed Recurrent Q-Networks* 2016. arXiv: `1602.02672 [cs.AI]`.

29. Foerster, J., Assael, I. A., De Freitas, N. & Whiteson, S. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems* **29** (2016).

30. Khetarpal, K., Riemer, M., Rish, I. & Precup, D. *Towards Continual Reinforcement Learning: A Review and Perspectives* 2022. arXiv: `2012.13490 [cs.LG]`.

31. Padakandla, S. A survey of reinforcement learning algorithms for dynamically varying environments. *ACM Computing Surveys (CSUR)* **54,** 1–25 (2021).

32. in. *Machine Learning Proceedings 1992* ().

33. Bengio, Y., Louradour, J., Collobert, R. & Weston, J. *Curriculum learning* in *Proceedings of the 26th annual international conference on machine learning* (2009), 41–48.

34. Selfridge, O. G., Sutton, R. S. & Barto, A. G. *Training and Tracking in Robotics.* in *Ijcai* (1985), 670–672.

35. Amodei, D. *et al. Deep speech 2: End-to-end speech recognition in english and mandarin* in *International conference on machine learning* (2016), 173–182.

36. Florensa, C., Held, D., Wulfmeier, M., Zhang, M. & Abbeel, P. *Reverse curriculum generation for reinforcement learning* in *Conference on robot learning* (2017), 482–495.

37. Soviany, P., Ionescu, R. T., Rota, P. & Sebe, N. Curriculum learning: A survey. *International Journal of Computer Vision* **130,** 1526–1565 (2022).

38. Narvekar, S. *et al.* Curriculum learning for reinforcement learning domains: A framework and survey. *The Journal of Machine Learning Research* **21,** 7382–7431 (2020).

39. Narvekar, S., Sinapov, J., Leonetti, M. & Stone, P. *Source task creation for curriculum learning* in *Proceedings of the 2016 international conference on autonomous agents & multiagent systems* (2016), 566–574.

40. Peng, B. *et al.* Curriculum design for machine learners in sequential decision tasks. *IEEE Transactions on Emerging Topics in Computational Intelligence* **2,** 268–277 (2018).

41. Silva, F. L. D. & Costa, A. H. R. *Object-oriented curriculum generation for reinforcement learning* in *Proceedings of the 17th international conference on autonomous agents and multiagent systems* (2018), 1026–1034.

42. Kemker, R., McClure, M., Abitino, A., Hayes, T. & Kanan, C. *Measuring catastrophic forgetting in neural networks* in *Proceedings of the AAAI conference on artificial intelligence* **32** (2018).

43. Matiisen, T., Oliver, A., Cohen, T. & Schulman, J. *Teacher-Student Curriculum Learning* 2017. arXiv: 1707.00183 [cs.LG].

44. Yu, J. & LaValle, S. *Structure and intractability of optimal multi-robot path planning on graphs* in *Proceedings of the AAAI Conference on Artificial Intelligence* **27** (2013), 1443–1449.

45. Ho, F. *et al.* Decentralized multi-agent path finding for UAV traffic management. *IEEE Transactions on Intelligent Transportation Systems* **23,** 997–1008 (2020).

46. Wen, L., Liu, Y. & Li, H. CL-MAPF: Multi-agent path finding for car-like robots with kinematic and spatiotemporal constraints. *Robotics and Autonomous Systems* **150,** 103997 (2022).

47. Lejeune, E., Sarkar, S. & Jezequel, L. *A survey of the multi-agent pathfinding problem* tech. rep. (Technical Report, Accessed July, 2021).

48. Li, J.-Q., Mirchandani, P. B. & Borenstein, D. The vehicle rescheduling problem: Model and algorithms. *Networks: An International Journal* **50,** 211–229 (2007).

49. Peer, E., Menkovski, V., Zhang, Y. & Lee, W.-J. *Shunting trains with deep reinforcement learning* in *2018 ieee international conference on systems, man, and cybernetics (smc)* (2018), 3063–3068.

50. O'Driscoll, N. Scaling reinforcement learning for the train shunting problem.

51. Alawad, H., Kaewunruen, S. & An, M. A deep learning approach towards railway safety risk assessment. *IEEE Access* **8,** 102811–102832 (2020).

52. Alawad, H., Kaewunruen, S. & An, M. Learning from accidents: Machine learning for safety at railway stations. *IEEE Access* **8,** 633–648 (2019).

53. Yang, C., Sun, Y., Ladubec, C. & Liu, Y. Developing machine learning-based models for railway inspection. *Applied Sciences* **11,** 13 (2020).

54. Hillier, F. S. & Liebermann, G. J. *Operations Research: Einführung* (Walter de Gruyter GmbH & Co KG, 2014).

55. *Flatland Evaluation Metrics* Accessed: 2024-01-20. https://www.aicrowd.com/challenges/flatland#evaluation-metrics.

56. *Flatland Introduction* Accessed: 2024-01-21. https://www.aicrowd.com/challenges/flatland#introduction.

57. *Flatland Introduction* https://blog.aicrowd.com/challenge-summary-flatland-48f4191c271a. Accessed: 2024-01-19.

58. *Flatland Winner of 2019 Interview* https://blog.aicrowd.com/flatland-challenge-winners-perspective-interview-with-mugurel-4a2988e67711. Accessed: 2024-01-25.

59. *Flatland Winner of 2020* https://discourse.aicrowd.com/t/neurips-2020-flatland-winners/4010?u=masterscrat. Accessed: 2024-01-25.

60. *Flatland Winner of 2021* https://www.aicrowd.com/challenges/flatland-3/winners. Accessed: 2024-01-25.

61. Li, J. *et al. Scalable rail planning and replanning: Winning the 2020 flatland challenge* in *Proceedings of the international conference on automated planning and scheduling* **31** (2021), 477–485.

62. Jiang, Y., Zhang, K., Li, Q., Chen, J. & Zhu, X. *Multi-Agent Path Finding via Tree LSTM* 2022. arXiv: `2210.12933 [cs.AI]`.

63. *Flatland Introduction* Accessed: 2024-01-21. `https://flatland.aicrowd.com/environment/observations.html`.

64. Auer, P., Cesa-Bianchi, N. & Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine learning* **47,** 235–256 (2002).

65. Mao, W., Zhang, K., Zhu, R., Simchi-Levi, D. & Basar, T. *Near-optimal model-free reinforcement learning in non-stationary episodic mdps* in *International Conference on Machine Learning* (2021), 7447–7458.

66. Kirkpatrick, J. *et al.* Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* **114,** 3521–3526 (2017).

67. Delfosse, Q., Schramowski, P., Mundt, M., Molina, A. & Kersting, K. Adaptive rational activations to boost deep reinforcement learning. *arXiv preprint arXiv:2102.09407* (2021).

68. Molina, A., Schramowski, P. & Kersting, K. Pad\'e Activation Units: End-to-end Learning of Flexible Activation Functions in Deep Networks. *arXiv preprint arXiv:1907.06732* (2019).

69. Leichthammer, L. Evaluating Planning-based Machine Learning Algorithms for Scheduling Railway Operations (2022).

Acknowledgements

I am grateful to Prof. Ramesh for providing me with the opportunity to pursue my thesis within his research group, where I gained valuable insights and was guided not to think about problems from just one angle but to also consider other views which exist apart from my own standpoint. Additionally, I appreciate the introduction to Achref Jaziri.
I extend my sincere gratitude to Achref Jaziri for his guidance and support throughout my research journey. Our extensive discussions on Reinforcement Learning and Machine Learning significantly contributed to my understanding of these complex fields.
My thanks also go to Maximilian Will and Felix Kreuer for proofreading and giving me feedback on this thesis.

Eigenständigkeitserklärung