KATHOLISCHE UNIVERSITÄT EICHSTÄTT-INGOLSTADT

Faculty of Economics Ingolstadt

Chair of Operations Management

Prof. Dr. Pirmin Fontaine

Introduction to Reinforcement Learning:

A Reinforcement Learning Approach for Train Rescheduling

Authors:	Gabriele Pfennig | Pavitra Chandarana | Lisa Peltier
Matriculation number:	11015954 | 202679 | 11011563
Submission date:	11.08.2024
Semester:	Sommersemester 2024
Location:	Ingolstadt
Examiner:	Prof. Dr. Pirmin Fontaine | Dr. Alexander Rave

Table of Contents

List of Figures

List of Tables

List of Abbreviations

DRL	Deep reinforcement learning
DQL	Deep Q-Learning
MDP	Markov Decision Process
km	Kilometers
km/h	Kilometers per hour
min	Minutes
RL	Reinforcement learning

List of symbols

A_t, a_t	Action at time t
$\leftarrow$	Assignment
γ	Discount factor
ε	Exploration rate (probability of performing a random action in ε-greedy policy)
R_t, r_t	Reward at time t
S_t, s_t	State at time t
α	Step size parameter
t	Discrete time step
$Q()$	Utility function

1. Introduction

In 2023, Deutsche Bahn faced significant challenges with punctuality, with nearly one-third of their customers experiencing delays in reaching their intended destinations (Tagesschau, 2023). This regular occurrence of unexpected delays highlights a critical issue regarding the reliability of railway systems. Train punctuality, defined as the difference between the actual and scheduled arrival times at the end destination, is the primary metric used to quantify reliability (Šemrov *et al.*, 2016, p. 250). To address these delays, it is necessary to handle the complex issue of train rescheduling, which involves making real-time decisions to minimize disruptions and stick to an optimal schedule, even when faced with unexpected changes to the schedule.

Train rescheduling is a dynamic procedure that requires dispatchers to react quickly to delays. They rely on their experience and intuition because it is unrealistic to fully investigate every potential approach to select the optimal solution given this limited timeframe (Hara *et al.*, 2006, p. 1). This emphasizes the need for smart decision-making tools that can quickly identify feasible options.

This report explores the application of reinforcement learning (RL) in train rescheduling. It starts by explaining the key concepts of RL, such as the Markov Decision Process, the base of the coding structure, Q-learning, and the ε-greedy Algorithm.

The first case will focus on a basic train scheduling scenario, addressing both a simple and an extended problem to demonstrate how train timetables can be optimized to minimize delays and improve overall efficiency. Following that, a more complex real-world scenario will be introduced, specifically analyzing the route from Ingolstadt to Munich, which involves the coordination of three different types of trains.

This research's objective is to demonstrate the application of RL in handling the complexities of train scheduling, to improve the reliability and punctuality of the railway system.

2. Reinforcement Learning in Train Scheduling

2.1. Markov Decision Process

The Markov Decision Process (MDP) framework is a powerful tool for applying RL in train scheduling. An MDP consists of an agent, states, actions, rewards, and the environment. *Figure 1* shows the interaction between these components. The illustration and explanation are based on the research of Sutton and Barto (1998).

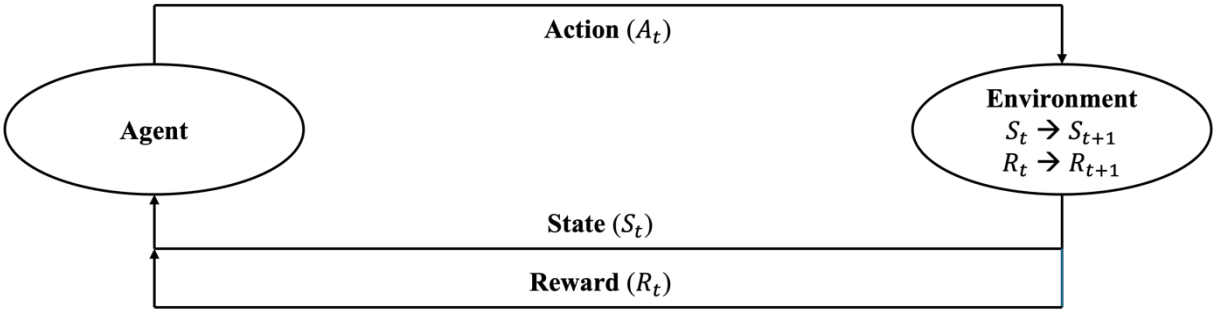


Figure 1: General Markov decision process in reinforcement learning (based on Sutton and Barto, 1998, p. 48)

The train settings in this framework are represented by the environment. It includes the timetable with departure and arrival times, the number of trains, the number of stops, the distance between stations, and the train velocities. The agent is responsible for making decisions about train movements. The action A_t refers to the decision made by the agent. Typically, this involves two choices: moving or waiting.

The state S_t represents the status of all trains on the track at a given time t which is specified in the timetable, all train positions and speeds, and other infrastructure components. After the action is executed, the environment transitions to a new state S_{t+1} at time $t + 1$ and all changes to the signal states resulting from the action carried out are considered.

The reward R_t quantifies the quality of the schedule at the time t. It calculates the benefit of being in a certain state or executing a specific action and provides feedback to the agent to help with its learning process. If the trains experience longer delays, the penalty increases. If the delays are shorter, the reward increases. This system incentivizes the agent to find ways to minimize delays and improve the schedule. At time $t + 1$, the agent is given a new reward R_{t+1}, which represents the outcome of the action taken in state S_t. This new reward helps the agent

evaluate the effectiveness of its previous action and adapts its strategy to maximize future rewards.

In conclusion, the MDP framework offers a systematic approach to train scheduling, allowing RL agents to make data-driven decisions that improve the efficiency and reliability of railway operations.

2.2. General Setting

To address the train scheduling problems outlined in this report, it is necessary to import different essential libraries. The libraries include the *gym* library, which is used to set up the customized environment, the *NumPy* library, which serves for numerical operations, and the *matplotlib* library, which helps to visualize the training process and its results.

With the relevant libraries in place, the next step is to create an environment that replicates train scheduling scenarios. First *TrainEnvironment* is defined as a custom environment, which incorporates multiple parameters and states, therefore creating a realistic setting for the application of RL algorithms. The *TrainEnvironment* contains various functions for initializing the environment, updating states, and simulating train movements.

Table 1 presents several functions which help the agent to manage and interact with the environment *TrainEnvironment* (Gymlibrary, 2022).

init	Implements the initialization process which establishes the initial parameters of the system. These parameters include stations, train types, positions, distances, speeds, and timetables.
reset	Restores the environment to its original condition. This involves placing all trains back at the starting station and establishing their initial times.
get_state_index	Retrieves the current state of the environment and converts it into a singular integer index, which is then used in the Q-table.
step	Executes the activities for the trains, updates their locations and calculates rewards based on their success.
render	Displays the positions of the trains for visualization purposes.

Table 1: Functions to manage the gym environment

After establishing the environment and its methodologies, the Q-learning algorithm is implemented. This is critical for training the agent to make optimal scheduling decisions depending on the given environment.

2.3. Q-Learning

The *q_learning* function is the central component in tackling train rescheduling problems as it defines the agent's decision strategy as a utility function $Q(s_t, a_t)$ (Šemrov *et al.*, 2016, p. 252). The process starts by initializing a Q-table, which is then updated iteratively utilizing the observed state-action pairs during training episodes. The Q-learning algorithm responds to the updated rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

where:

- α is the step size parameter,
- γ is the discount factor,
- r_{t+1} is the reward received after taking an action a_t in state s_t,
- $Q(s_t, a_t)$ and $Q(s_{t+1}, a_{t+1})$ are the values of the Q-function for the respective states and actions.

The agent uses a comparison between a probability and ε to determine whether to engage in random exploration of actions or to exploit the action that is now considered the most optimal. The *ε-greedy* approach guarantees a trade-off between the exploration of new actions and the exploitation of learned actions. The Q-table is updated by applying the Bellman equation, which considers both the immediate benefit and the maximum expected future reward. This repeated approach persists until the agent acquires an optimal policy for train scheduling (Sutton and Barto, 1998, p. 131; Bulut, 2022, pp. 3–4). The *test_agent* function is used to see how well the trained agent does by running the environment with the trained Q-table and choosing actions based on the learned policy.

2.4. ε- Greedy Algorithm

The ε-greedy approach is applied to improve the learning efficiency of the Q-learning algorithm. The agent decides whether to participate in random action exploration or to exploit the action that is now deemed the most optimal by comparing a probability and a ε value. The ε-greedy strategy ensures a trade-off between exploring new actions and exploiting previously learned actions (Sutton and Barto, 1998, p. 28; Deeplizard, 2022). In our approach, ε is fixed at 0.1, resulting in a 10% chance of investigating a random action and a 90% chance of exploiting the best-known action. This steady degree of exploration enables the agent to balance learning new actions with refining its present approach, avoiding it becoming overly greedy and potentially resulting in suboptimal policies.

2.5. Testing

After training the Q-learning algorithm, it is crucial to assess its performance. The *test_agent* function examines the trained Q-table by running the environment and choosing actions based on the learned policy. This function replicates test episodes to see how well the trained agent performs in each episode, providing a practical assessment of the agent's decision-making abilities.

2.6. Case Assumptions

There are basic assumptions we took for setting the environment:

1) All trains are considered as a point object, i.e., every train travels through the block sections sequentially; to enter a block section, one must first exit the preceding block section.
2) To ensure safety, we follow the following fundamental rules: a train can only be in a block section at one time; a train in motion can only stop at stations, never in the middle of the block section; and a train can move on to the next block section when it is free, meaning that no other trains are on that section at that time.

3. Basic Train Scheduling Problem

This chapter describes our approach to solving a simple and complex train scheduling problem with RL. The process consists of creating a customized environment using OpenAI's Gym framework, applying a Q-learning algorithm, and assessing the results using various visualizations. The goal is to train an agent who can make accurate decisions under time constraints.

3.1. Environment Setup

The environment for the simple problem consists of three trains (0, 1, 2) and three railway stations (A, B, C). Train positions are set to zero in an array, indicating that all trains begin at the first station. Distances between the stations are predetermined in kilometers: A-B: 8.8 km and B-C: 5.8 km. The average speeds of the trains are specified in kilometers per hour (km/h) and reach 70km/h, for train 0 and 100 km/h for train 1 and 2.

The action space is defined as a multi-discrete space in which action = 1 means “move to next station” and action = 0 means “stay”.

Rewards and penalties are determined by the punctuality of the trains:

- An incentive of +10 is provided for punctual or early attendance.
- A deduction of -10 is imposed for delays.
- There is a penalty of -5 for waiting.

Q-learning training is performed for 5000 episodes. The Q-learning algorithm uses the following parameters:

- Learning rate (α): 0.1
- Discount factor (γ): 0.9
- Exploration rate (ε): initially set to 1 and gradually decreases to 0.01 over time with decay of 0.001. The gradual decline leads the agent´s strategy to transition from exploration to exploitation.

3.2. Results

Figure 2 shows the cumulative reward earned across number of episodes throughout the training phase. In the first 1,000 training episodes, the rewards are highly variable, with many

episodes resulting in negative rewards. This indicates that the agent is exploring different actions and often making suboptimal decisions with overall lower rewards. After approximately 2,000 episodes, the rewards begin to stabilize, with fewer extreme fluctuations. The rewards settle at a value of around 40, indicating that the agent has learned an effective strategy to optimize train scheduling. Nevertheless, the occasional decreases in reward indicate that there are still suboptimal choices being made, potentially due to the exploration component of the algorithm.

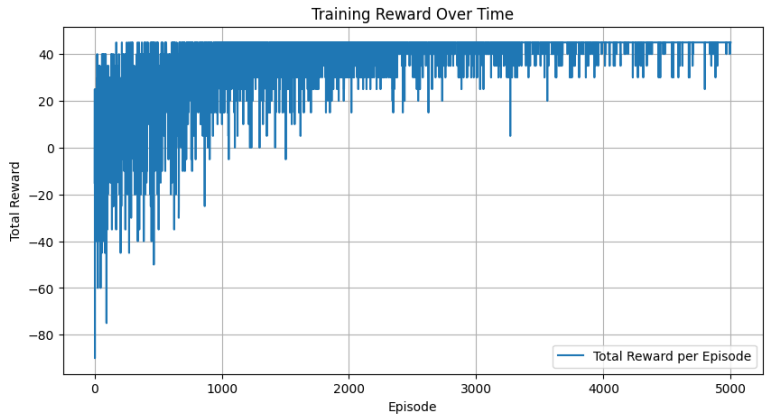


Figure 2: Training rewards earned across 5,000 training episodes

Regarding the train positions over time, *Figure 3* displays the positions of the trains at each step throughout a test episode. The trains travel from station 0 (A) to station 3 (None), with occasional delays seen in the intermediate stages. This indicates that the trains are effectively progressing towards their intended destinations. The incremental steps in the decision-making process are represented by the delays or periods when trains remain stationary. These pauses may reflect the train waiting for track clearance or deliberately staying in position as part of a strategic choice.

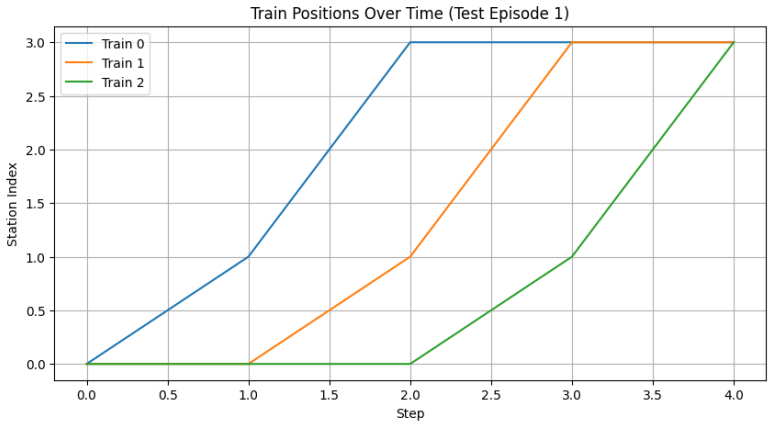


Figure 3: Train position across steps in test episode

Additionally, *Appendix A, Figure 1* shows a heatmap displaying the distribution of Q-values, which indicates the agent's assessment of the rewards is associated with various actions in state 0. Greater Q-values indicate that the agent perceives these activities as more beneficial.

The findings of this basic model indicate that the agent can learn the skill of making smart scheduling decisions. However, the simplicity of the model and the inherent constraints of the Q-learning algorithm in a discrete, limited-scale environment highlight the need for more advanced approaches and concerns to be implemented in actual, real-world applications.

3.3. Extension of Basic Problem

The basic Q-learning approach tends to overstate the action values by selecting and evaluating actions based on the same Q-values. Overestimation can result in less efficient solutions, especially in complex scenarios with many state-action combinations. In the extension, the agent has much more control, which might lead to collisions due to the agent making drastic decisions.

Environment Setup

The state space features a variety of train locations and speeds. Locations include 'Platform A', 'Platform B', 'On way to B', and 'Station B'. Speeds range from 50 to 300 km/h, with 50 km/h increments50 km/h.

The action space has six possible actions: 'Depart', 'Arrive', 'Wait', 'Speed Up', 'Slow Down', and 'Maintain Speed'.

Rewards and penalties are determined as follows:

- Collisions have a substantial penalty of -5,000.
- A generous reward of +10,000 is offered for successfully reaching Station B by following the route 'On route to B'.
- Penalties are -1 for 'Depart', -5 for 'Wait', and -2 for 'Speed Up' and 'Slow Down'.
- 'Maintain Speed' action does not provide any reward.

The state of the environment is updated under the selected actions. When performing 'Speed Up' and 'Slow Down' actions, the speed changes are limited to the set range. Location transitions are specifically designated for the acts of 'Depart' and 'Arrive', facilitating the movement of trains between platforms, routes, and stations.

The environment includes features for determining whether a train has reached a terminal state, specifically if it has arrived at 'Station B'. In addition, a mechanism is created to identify collisions between trains running on the same route, guaranteeing that the simulation appropriately accounts for any potential conflicts caused by train movements.

Furthermore, collisions are monitored and tracked for a duration of 120 seconds during the simulation.

Q-learning training is performed for a total of 10,000 episodes. The Q-learning algorithm uses the next parameters:

- Learning rate (α): 0.1
- Discount factor (γ): 0.9
- Exploration rate (ε): initially set to 1 and gradually decreases to 0.01 over time with decay of (initial epsilon – final epsilon / number of episodes).

Results

Concerning the training rewards over time, *Appendix A, Figures 2 & 3* show the reward history for both trains demonstrate a trend in learning. Initially, the rewards are strongly negative, which indicates a poor level of performance characterized by frequent collisions and poor decision-making. Over time, the rewards for both trains gradually increase, suggesting a significant improvement in performance. This upward trend indicates that the agents are developing the ability to optimize their actions, minimize collisions, and achieve their goals more efficiently. However, the fact that there are still shifts in future episodes suggests that although the agents have made progress, they are still in the process of exploring and sometimes making suboptimal decisions, which is expected given the ε-greedy approach.

Examining the collision history, *Figure 4* illustrates the frequency of collisions throughout the training process. Initially, many collisions indicate that the agents regularly make suboptimal choices because they are still adopting effective strategies. As the training progresses, there is a significant decrease in collisions. By the end of the training period, the frequency of collisions has nearly reached zero, suggesting that the agents have effectively acquired the ability to avoid dangerous actions and enhance their coordination.

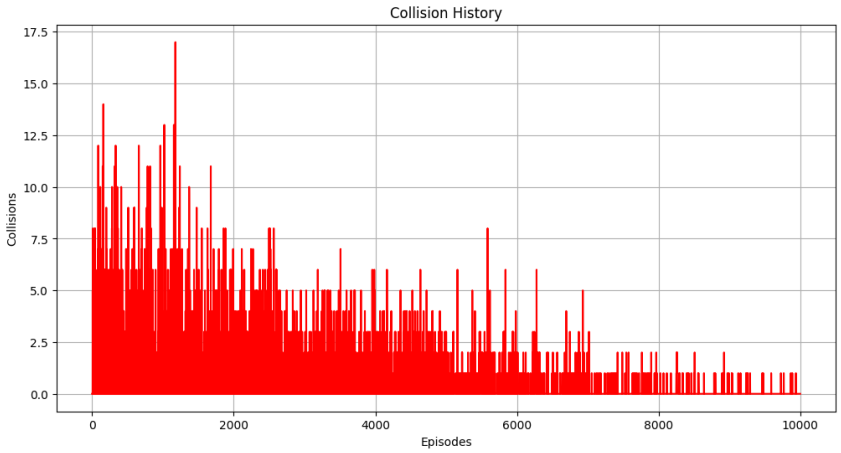


Figure 4: Collision history throughout the training process

Appendix A, Figure 4 shows the rate at which collisions decrease over each episode. The significant decrease in collisions during the initial moments of each episode indicates how quickly the agents learn from their experiences and adapt their strategies in response. The extremely low number of collisions observed at the end of the 120-second window in each episode demonstrates successful learning and the implementation of highly efficient collision avoidance strategies. In addition, the testing process reveals no collisions across all test episodes, confirming the robustness of the learned strategies.

Overall, the results suggest that the Q-learning agents successfully develop the ability to optimize train scheduling decisions over time. The upward trend in rewards, decreasing collisions, and stabilizing Q-values indicate successful learning and enhanced performance. The agents' consistent choice of high-reward actions and their balanced exploration of many strategies suggest that they are making informed and rational decisions. Despite the initial challenges, the agents make significant progress during both the training and testing processes, demonstrating the potential of Q-learning in tackling complex train scheduling issues. A possible improvement of the model could entail the implementation of advanced algorithms and the consideration of practical concerns to improve performance and robustness.

4. Complex Train Scheduling Problem

This chapter focuses on a complex environment for the Q-learning model. It includes the integration of actual train schedules (Deutsche Bahn, 2024) to create a realistic simulation that represents the complexity of optimizing train movements and timetables. In this section, additional variables and constraints are introduced that accurately represent the operational challenges associated with different train types and their varying schedules. The inclusion of these elements provides a comprehensive understanding of the intricacies of managing complex train schedules.

4.1. Environment Setup

The environment in the complex train scheduling includes two types of trains (RB 16 and ICE) operating across nine stations: Ingolstadt, Baar-Ebenhausen, Rohrbach, Pfaffenhofen, Reichertshausen, Paindorf, Petershausen, Dachau, and München. The RE 1, which usually operates with the other two train types on the route, is excluded in our case due to cancellations.

Table 2 shows the distances between consecutive stations defined in km and the travel times of the respective train types in minutes (min). The RB 16 travels at an average speed of 90.3 km/h, while ICE travels at 135.6 km/h.

From – to section	Distance	RB 16 section time	ICE section time
Ingolstadt to Baar-Ebenhausen	9.5 km	9 min	6 min
Baar-Ebenhausen to Rohrbach	14 km	8 min	5 min
Rohrbach to Pfaffenhofen	11.5 km	7 min	5 min
Pfaffenhofen to Reichertshausen	5.9 km	5 min	3 min
Reichertshausen to Paindorf	3.5 km	4 min	3 min
Paindorf to Petershausen	3.5 km	4 min	3 min
Petershausen to Dachau	18 km	10 min	6 min
Dachau to München	20 km	12 min	8 min

Table 2: Distance between stations and travel times of RB 16 and ICE

The action space is defined similarly to the basic train scheduling problem in *part 3.1.* (1 means "move to the next station"; 0 means "stay at the current station.").

The reward system and penalties are based on the trains' punctuality. At each station, the delay is calculated based on the actual arrival time compared to the planned time. The corresponding penalty is set as follows:

- Based on delay intervals of 5 minutes with a penalty-range from -1 to -20 (larger delays lead to a higher penalties)
- An additional penalty of -10 is applied for attempting to move to an already occupied track.
- A penalty of -10 is imposed for not moving.

Q-learning, with a total of 5,000 episodes, has the same parameters as the simple train scheduling problem:

- Learning rate (α): 0.1
- Discount factor (γ): 0.9
- Exploration rate (ε): initially set to 1 and gradually decreases to 0.01 over time with decay of 0.001.

4.2. Results

Figure 5 represents the total rewards accumulated per episode over 5,000 training episodes in the Q-learning process.

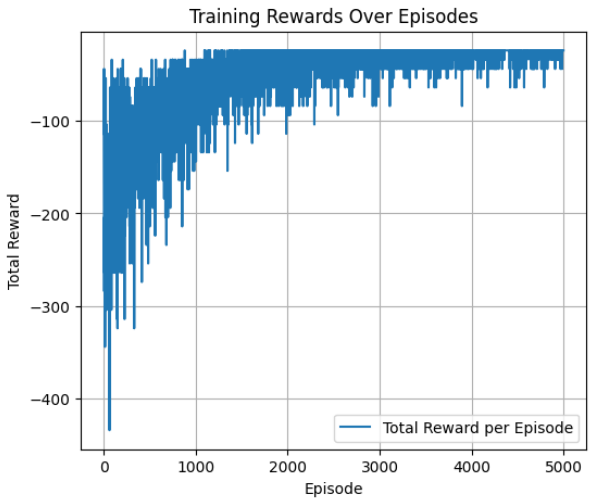


Figure 5: Training rewards over the period of 5,000 training episodes

In the first 500 episodes, there is high variability in rewards which indicates that the agent is initially exploring and frequently making suboptimal decisions, resulting in significant penalties. As training progresses the total rewards start to increase steadily as the model is

learning from its mistakes and gradually improving its policy. After approximately 2,000 episodes, the rewards become more stable and settle at a higher value of around -100. This stabilization indicates that the agent has learned an efficient strategy, although it still receives punishments due to the defined environment and reward system.

Figure 6 represents the positions of two types of trains, ICE and RB 16, over time during one test episode. The station index shows how far the train is along the route (0 = Ingolstadt; 8 = München).

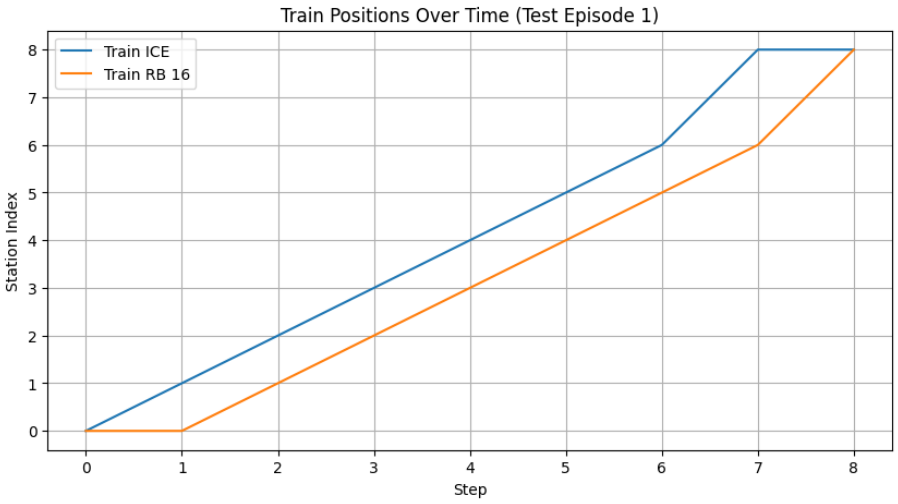


Figure 6: Train positions of RB 16 and ICE after one test episode

Both trains start at the initial station (station index 0). The ICE (blue line) starts moving right from step 1 and consistently advances, with some variation in its rate of progress. It each station ahead of the RB 16 (orange line) which indicates that ICE has a higher priority and is faster according to the model.

Overall, the results of the complex problem imply that the Q-learning model effectively learns to manage train scheduling by improving decision-making over time. The model demonstrates its ability to prioritize faster trains, as seen with the ICE consistently moving ahead of the RB 16 in the test episode. However, the persistence of penalties suggests that there is still room for further refinement in the scheduling strategy to minimize delays and optimize performance.

5. Discussion on Performance and Limitations of the RL-Models

All models allow the agent to adapt to the environment and make the most of it. The number of penalties decreases significantly over the training episodes in the first 2,000 episodes and then gradually converge to the optimal value. Ideally, as the model learns more effective rules, the rewards should either grow, decrease or remain constant. This situation can be seen in all models, suggesting minimized instability.

While the complex model appears to be promising in handling complex train schedules, complexities such as train length, number of trains, and complex railway structure can be difficult to overcome. To address such issues, techniques such as deep q-learning (DQL) and deep reinforcement learning (DRL) can be used to improve its efficiency and accuracy in a real-world setting.

Modifying the training parameters, such as the learning rate, discount factor, and exploration rate, might help in further stabilizing the learning process. Increasing the rate at which exploration decreases could enable the model to utilize acquired techniques sooner, potentially decreasing the fluctuation in rewards.

To strengthen the models, one could consider expanding the scope of the environment, integrating more complex reward systems, and studying strategies involving several agents. For instance, by including several types of actions, such as modifying velocity instead of only moving or remaining stationary, the agent can have greater control and make more effective scheduling decisions.

By incorporating further factors such as varying time intervals, new train types, and their corresponding delays, a better comprehension of the scheduling difficulties can be achieved. Nevertheless, this would require the redefinition of the observation space to effectively handle the increased complexity.

6. Summary

When it comes to the performance across models, Q-learning demonstrated good performance in the basic problem, with a noticeable enhancement in rewards as time progressed. The rewards progressively grew as the model acquired the ability to optimize its activities, resulting in finding the optimal Q-table. Furthermore, in the extension model, the initial reward suggests that the model was experimenting with various approaches, frequently resulting in poor choices. Nevertheless, as the training progressed, these variations decreased in intensity, and the general trend of rewards exhibited substantial enhancement. The decrease in the occurrence and intensity of collisions over time emphasized the model's ability to adjust to complexity. The gradual decrease in the reward states that the model indeed finds the optimal policy that can be relied on. Finally, several realistic factors and constraints were included in this complex train scheduling problem, including the actual train schedules and the different types of trains (RB 16 and ICE).

Despite these difficulties, the model showed remarkable performance by successfully navigating this complex environment and reliably determining the best course of action. The model rapidly improves its performance and stabilizes near-ideal rewards, as evidenced by the training rewards over episodes, which show a significant improvement as the training advances. This pattern illustrates the model's capacity for efficient learning and adaptation, which leads to dependable and successful solutions despite the task's complexity. The findings of these experiments show that, while Q-learning can tackle train scheduling challenges, more improvements and optimizations are required to generate more reliable and efficient solutions for real-world application.

Appendices

Appendix A: Basic Train Scheduling Problem

Figure 1: Heatmap for Q-values associated with state 0 (simple problem)

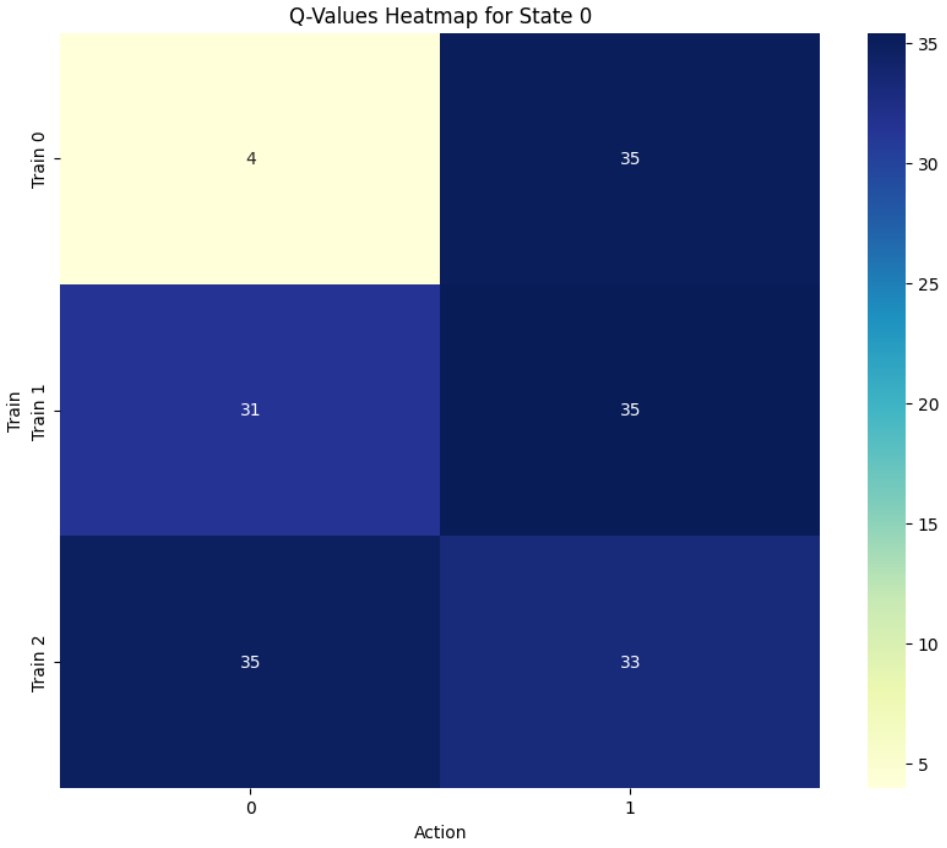


Figure 2: Reward History for Train 1 (extension of basic problem)

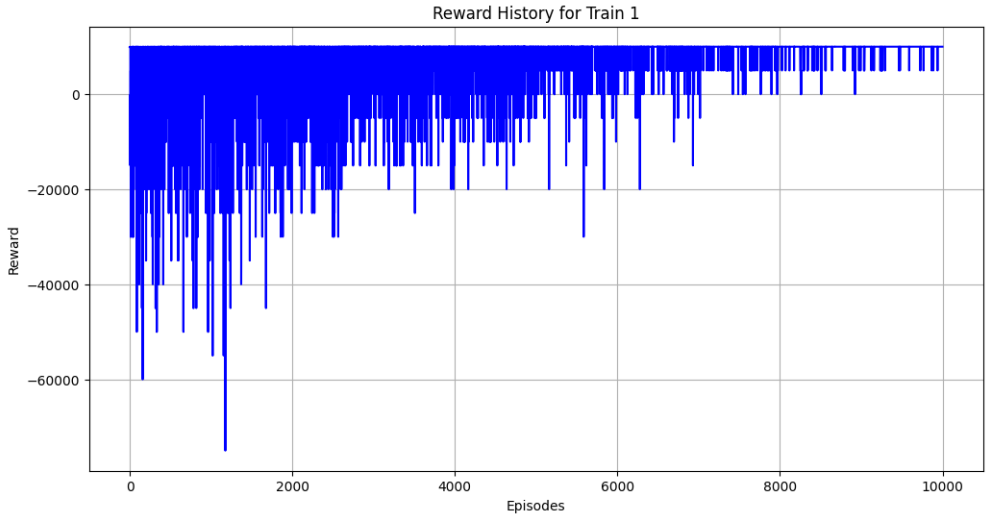


Figure 3: Reward History for Train 2 (extension of basic problem)

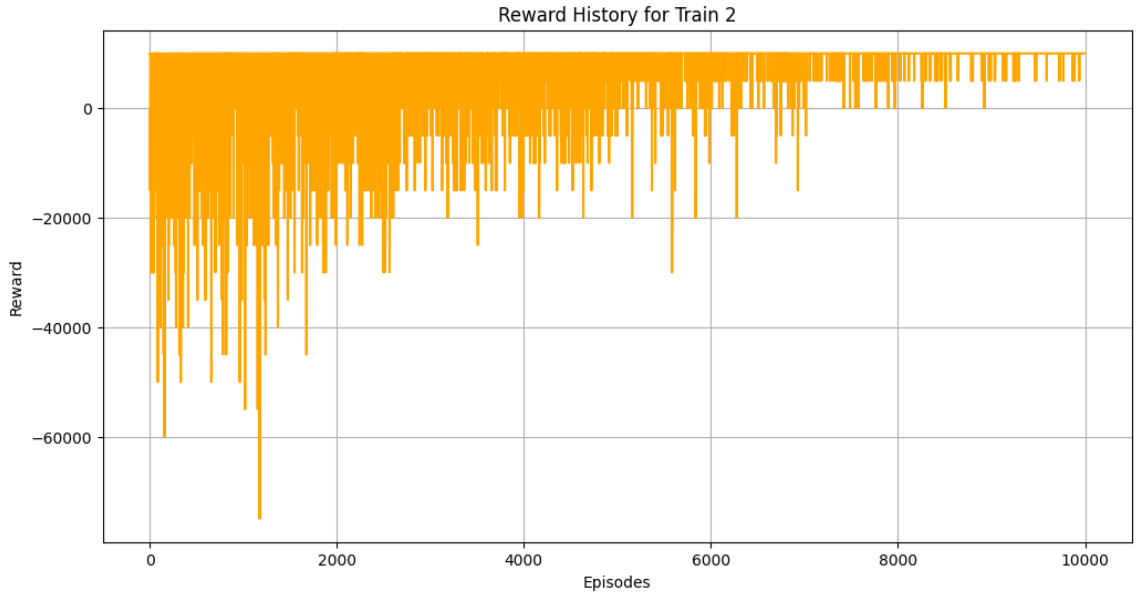
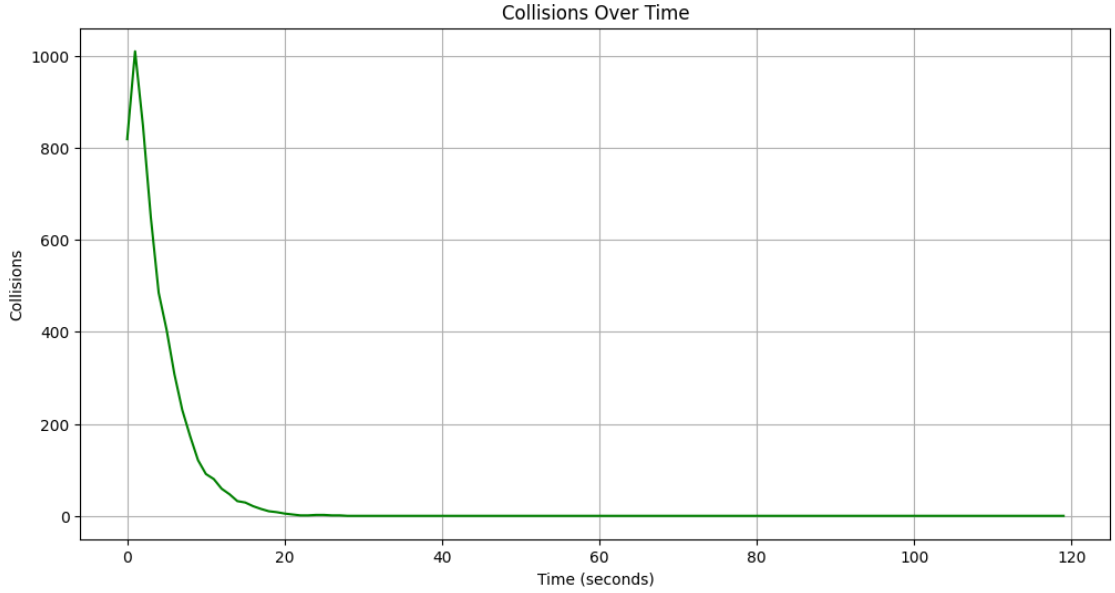


Figure 4: Collisions over time (extension of basic problem)



List of References

Bulut, V. (2022) ‘Optimal path planning method based on epsilon-greedy Q-learning algorithm’, *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 44(3), p. 106. Available at: https://doi.org/10.1007/s40430-022-03399-w.

Deeplizard (2022) ‘Reinforcement Learning - Developing Intelligent Agents’. Available at: https://deeplizard.com/learn/video/HGeI30uATws.

Deutsche Bahn (2024) ‘Deutsche Bahn Fahrplan’. Available at: https://www.bahn.de.

Gymlibrary (2022) ‘Gym Documentation - Make your own custom environment’. Available at: https://www.gymlibrary.dev/content/environment_creation/.

Hara, K., Kumazawa, K. and Koseki, T. (2006) ‘Efficient Algorithm for Evaluating and Optimizing Train Reschedules by Taking Advantage of Flexibility of Quadruple Track’.

Šemrov, D., Marsetič, R., Žura, M., Todorovski, L. and Srdic, A. (2016) ‘Reinforcement learning approach for train rescheduling on a single-track railway’, *Transportation Research Part B: Methodological*, 86, pp. 250–267. Available at: https://doi.org/10.1016/j.trb.2016.01.004.

Sutton, R.S. and Barto, A.G. (1998) *Reinforcement learning: an introduction*. Cambridge, Mass: MIT Press (Adaptive computation and machine learning).

Tagesschau (2023) ‘Fast jeder dritte Reisende kam 2023 verspätet an’. Available at: https://www.tagesschau.de/wirtschaft/unternehmen/deutsche-bahn-verspaetung-100.html.

Statutory Declaration

We herewith declare that we have composed the present thesis ourselves and without the use of any other than the cited sources and aids. Sentences or parts of sentences quoted are marked as such; other references concerning the statement and scope are indicated by full details of the publications concerned. The thesis in the same or similar form has not been submitted to any examination body and has not been published.

Ingolstadt, 11.08.2024

Gabriele Pfennig

Pavitra Chandarana

Lisa Peltier