Existing and novel Approaches to the Vehicle Rescheduling Problem (VRSP)

In the course of the Flatland Challenge by Swiss Federal Railways (SBB)

Jonas Wälter

University of Applied Sciences Rapperswil (HSR)

January 26, 2020

Rapperswil, Switzerland

Supervisor:	Prof. Dr. Farhad D. Mehta
External examiner:	Dr. Xiaolu Rao

Abstract

The demands on railway transport are constantly increasing and the transport capacity is reaching its limits. An opportunity to address this challenge is the optimization of traffic planning through novel approaches. In this report, existing and novel approaches to two fundamental problems in transport planning are examined with the help of a framework provided by Swiss Federal Railways (SBB).

For the Multi-Agent Path Finding (MAPF) problem, there are different optimal approaches such as Linear Programming, Constraint-based Search, and Operator Decomposition & Independence Detection, as well as suboptimal approaches like Prioritized Planning. The optimal algorithms provide an optimal solution for the problem, but at the expense of a non-polynomial runtime. Instead, a suboptimal approach is to be chosen which provides an appropriate solution within a polynomial runtime.

For the Vehicle Rescheduling Problem (VRSP), various approaches are investigated as well. A special focus is given to the novel Reinforcement Learning approach, where a train should learn independently by means of artificial intelligence how to behave best in a situation. In this report, the potential of reinforcement learning is demonstrated, as it can almost keep up with the existing approaches.

An additional challenge is the detection of deadlocks, where several trains inextricably block each other. A complete and reliable detection is again not achievable within a polynomial runtime. Instead, simplified algorithms or again reinforcement learning can be used for a faster but incomplete detection of deadlocks.

In conclusion, it can be stated that reinforcement learning is a novel but promising method, which could be used in railway traffic in an optimizing way. This approach may not be able to replace the existing traffic management methods, but it can at least support these methods.

Executive Summary

Introduction

Every day, vast quantities of passengers and freight are transported around the globe. Since the demand for mobility is growing, the transport capacity will have to be massively increased in the future. The Swiss Federal Railways (SBB), the national railway company of Switzerland, would like to investigate novel approaches to optimize traffic management. For this reason, a public competition called Flatland Challenge was launched.

Problem

The Flatland Challenge aims to address two fundamental problems in the world of traffic. The first problem concerns Multi-Agent Path Finding (MAPF), where any number of trains should be routed from their current location to the desired destination as fast as possible but without any conflicts. However, a train can unexpectedly be disturbed, for example by a malfunction. In such a situation, the Vehicle Rescheduling Problem (VRSP) occurs, where the malfunction causes a replanning of the routes.

Figure 1: Railway network with multiple operating trains

Approaches

For these problems, there are several existing approaches, as well as novel methods. In the course of the thesis, the following approaches to the MAPF problem were investigated:

- Linear Programming
- Constraint-based Search
- Operator Decomposition & Independence Detection
- Optimal Anytime Algorithm
- Prioritized Planning

These approaches can be divided into two categories: First, there are algorithms finding an optimal solution for each situation. However, this entails a significant additional expenditure of calculation time. Thus, there are also algorithms finding a solution within a reasonable time, but this solution is not necessarily optimal.

Furthermore, different approaches for the VRSP problem were examined: In principle, the remaining routes can be rescheduled using a MAPF algorithm in case of a malfunction. An alternative approach is to plan the routes from the beginning in a robust way that they keep functioning in case of a malfunction. A special focus was given to the novel Reinforcement Learning approach, where a train should learn independently by means of artificial intelligence how to behave best in a situation.

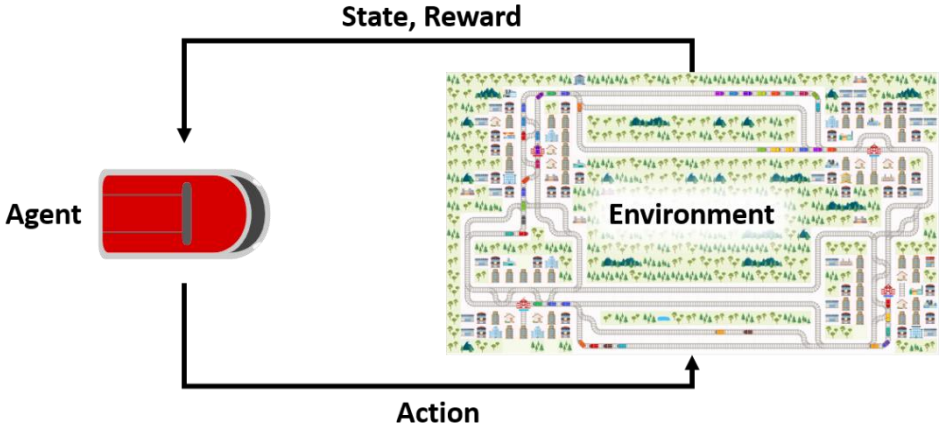


Figure 2: Reinforcement learning: scheme of the learning cycle

An additional risk in a railway network is the occurrence of deadlocks, where several trains block each other. Therefore, various approaches to detect deadlocks were considered as well.

Results

All approaches were implemented and evaluated. Concerning MAPF, it was shown that the theoretically optimal solutions were not suitable for use in a real-time traffic management system. Instead, suboptimal algorithms as prioritized planning that provide fast solutions are more appropriate. For VRSP, reinforcement learning was almost able to keep up with the results of the other approaches and thus demonstrated the potential of this novel approach.

Reinforcement learning is still a very young and little researched area. There are many more optimization possibilities and the results could certainly be improved with additional effort. In conclusion, reinforcement learning is a novel but promising method, which could be used in railway traffic in an optimizing way.

Table of Contents

Table of Figures

Table of Tables

Table of Abbreviations

2D	two-dimensional
BFS	Breadth-first search
CBC	Coin-or branch and cut
CBS	Constraint-based search
CPR	Complete path reservation
DRR	Dynamic route reservation
e.g.	for example
etc.	etcetera
i.e.	id est
ID	Independence detection
ILP	Integer linear programming
km	kilometers
LP	Linear programming
MAPF	Multi-agent path finding
MCA	Movement consequence analysis
MGS	Maximum group size
MILP	Mixed integer linear programming
NP	Nondeterministic polynomial time
OAA	Optimal anytime algorithm
OD	Operator decomposition
PP	Prioritized planning
RGB	red, green, blue
RL	Reinforcement learning
SBB	Swiss Federal Railways
SPP	Shortest path problem
TMS	Traffic management system
VBTC	Vehicle-based train control
VRSP	Vehicle rescheduling problem
VSP	Vehicle scheduling problem

1 Introduction

Every day, vast quantities of passengers and freight are transported around the globe. Since the demand for mobility is growing, the transport capacity will have to be massively increased in the future. There are different approaches to achieve this. One promising approach is the optimization and automation of traffic management systems.

A research group of the *Swiss Federal Railways (SBB)* wanted to investigate this approach in more detail and therefore launched a public competition called *Flatland Challenge*. In principle, the Flatland Challenge addresses two fundamental problems in the world of traffic. The first problem concerns *Multi-Agent Path Finding (MAPF)*, where any number of trains should be routed from their current location to their desired destination in a railway network. A solution is to be found for this in which all trains reach their destination in the shortest possible time. Thereby, it is also important to avoid conflicts between several trains for a feasible solution. However, a train can unexpectedly be disturbed, for example by a malfunction. In such a situation, the *Vehicle Rescheduling Problem (VRSP)* occurs, where the malfunction causes a replanning of the routes.

The Flatland Challenge is taken as an opportunity to explore and examine different approaches to these underlying problems of traffic control. The aim is to consider both already known, existing approaches and novel approaches such as Reinforcement Learning. The corresponding approaches for MAPF and VRSP are investigated in the course of the participation in the Flatland Challenge.

This report is organized as follows: Section 2 contains the background information on the Flatland Challenge, which serves as a basis for the other parts of the report. In Section 3, the given problem is analyzed, especially the requirements and the domain of the existing Flatland Environment. In Section 4, the introduction to traffic planning is provided by developing the encoding of the railway network and by the general search for the shortest path. Subsequently, the different approaches to the MAPF problem and VRSP are considered. The MAPF problem as part of Round 1 of the Flatland Challenge is covered in Section 5. Round 2 of the Flatland Challenge with VRSP as the central focus is covered in Section 6. The results of these approaches are then reviewed and discussed in Section 7. Finally, the report is closed with a conclusion in Section 8.

2 Background

2.1 Swiss Federal Railways (SBB)

The Swiss Federal Railways (SBB) operate as the national railway company of Switzerland and maintain and operate the largest railway infrastructure in the country.[1] The SBB are responsible for around 800 rail stations and over 3,200 km of managed routes with around 35,000 train signals. More than 10,000 trains travel every day, carrying 1.25 million passengers and 205,000 tons of freight to their destination.[2]

2.2 Traffic Management System (TMS)

The SBB are investigating various approaches to automated *Traffic Management Systems (TMS)*. The task of a TMS is to select the routes for all trains and set their priorities at the switches in order to optimize traffic flow throughout the railway network. The main objective is for all trains to arrive at their destination with a minimum travel time respectively delay.

2.3 Flatland Challenge

The SBB launched the Flatland Challenge as a public competition to tackle the VRSP. In the course of the Flatland Challenge, various new approaches and solutions could be found, which may affect the implementation of modern traffic management systems - not only in railway but also in other transport or logistics areas. The Flatland Challenge is oriented towards approaches in the field of Reinforcement Learning (RL). However, the Flatland Challenge is also open to other approaches.

An overview of the Flatland Challenge is available on the official competition website.[3] In the following subsections, the general information is summarized.

Figure 3: Flatland Environment with a railway network and multiple operating trains

[1] (We are SBB, 2019)

[2] (SBB Facts and Figures, 2019)

[3] https://www.aicrowd.com/challenges/flatland-challenge

2.3.1 Flatland Environment

The SBB has developed an environment that can simulate an arbitrary railway network and its trains in a simplified way. This environment is built as a two-dimensional grid environment with restricted transitions between adjacent cells for the representation and simulation of railway networks. The railway network can be of any size and contain any number of trains. An example of such a Flatland Environment with a railway network and several trains is shown in Figure 3.

A software library written in Python is provided by SBB for the Flatland Challenge, with which such environments can be simulated and displayed. All figures in this report of such railway networks or parts of them are generated using this library. The detailed domain analysis of the library and the Flatland Environment is described in Section 3.2.

2.3.2 Schedule

The Flatland Challenge consists of three consecutive rounds with increasing difficulty, which can be briefly described as follows:

Round 0: Learn to navigate
: There is a railway network with a single train that has an assigned start position and an assigned target position. The task is to find the shortest path for the train through the railway network.

Round: 1: Avoid conflicts
: The situation from Round 0 is extended, so that not only one train but any number of trains are present in the railway network. Thus, conflicts between the trains can occur, which are to be avoided.

Round 2: Optimize train traffic
: The situation from Round 1 is extended, so that the trains have different speeds and malfunctions can occur stochastically. In this case, a train is defective for a stochastic period of time.

3 Problem Analysis

Before starting to explore the different approaches to the problem, the actual problem has to be analyzed in detail. The problem definition is mostly given by the tasks and the environment of the Flatland Challenge.

3.1 Requirements Analysis

Based on the given tasks of the Flatland Challenge, the functional requirements can be analyzed and specified. The description of the tasks is carried out in the form of *Use Cases* in fully dressed format according to (Larman, 2005).

3.1.1 Actors

The Flatland Environment contains only one actor type, which is simply a train. For a more general description, the term *Agent* will be used for a train.

3.1.2 Use Cases

An agent as the only actor in the system has one single task to complete. This task is basically similar for all rounds of the Flatland Challenge. Round 2 is slightly different regarding the placement of the agent at the start position and its behavior at the target. Therefore, the common use case for Round 0 and Round 1 is listed in Table 1 and that for Round 2 in Table 2 separately.

Table 1: Use case for Round 0 and Round 1

Use Case	Navigate to target
Goal	The agent navigates in the railway network from its assigned start position to its assigned target position.
Primary Actor	Agent
Pre-conditions	• Flatland Environment exists • Agent is placed on a valid start cell in the railway network • Valid target position in the railway network is assigned to the agent • A maximum number of time steps is defined
Post-conditions	• Agent has reached its assigned target in the railway network • As few time steps as possible have been used for all agents to reach their target (minimization of combined travel time)
Main Success Scenario	1. The agent is located on its assigned start cell in the grid of the Flatland Environment. 2. The agent decides which movement action (move forward, move left, move right, stop) to perform next. 3. The agent executes the chosen action and thereby reaches a next cell in the grid. If the agent wants to execute an action that is not allowed on the current cell, it remains on the same cell. 4. The agent repeats steps 2-3 until it has reached its target.
Extension	• If the defined time limit is reached, the simulation is aborted, even if the agent has not yet reached its target. • If a conflict has occurred and agents block each other, the agents cannot reach their target. This leads to an endless loop (steps 2-3), which can only be resolved by aborting the simulation.

Table 2: Use case for Round 2

Use Case	Navigate to target
Goal	The agent navigates in the railway network from its assigned start position to its assigned target position.
Primary Actor	Agent
Pre-conditions	• Flatland Environment exists • Valid start cell in the railway network is assigned to the agent • Valid target position in the railway network is assigned to the agent • A maximum number of time steps is defined
Post-conditions	• Agent has reached its assigned target in the railway network • Agent has been removed from the railway network • As few time steps as possible have been used for all agents to reach their target (minimization of combined travel time)
Main Success Scenario	1. The agent waits for an arbitrary amount of time if desired. 2. The agent enters the Flatland Environment and then stands on its assigned start cell in the grid. 3. The agent decides which movement action (move forward, move left, move right, stop) to perform next. 4. The agent executes the chosen action and thereby reaches a next cell in the grid. If the agent wants to execute an action that is not allowed on the current cell, it remains on the same cell. 5. The agent repeats steps 3-4 until he has reached his target. 6. The agent leaves the railway network again.
Extension	• If the defined time limit is reached, the simulation is aborted, even if the agent has not yet reached its target. • If a conflict has occurred and agents block each other, the agents cannot reach their target. This leads to an endless loop (steps 3-4), which can only be resolved by aborting the simulation.

As a comparison of both tables shows, an agent in Round 2 is not placed on a cell in the grid from the beginning but has to enter the railway network first with an action. Thus, several agents may have been assigned the same start position in Round 2. This means that the solution approach must additionally decide in which order the agents should enter the railway network.

3.2 Domain Analysis

In the course of the domain analysis, the Flatland Environment provided by SBB is analyzed. In a simplified form, there are three important concepts, which are shown in the domain model in Figure 4. In the following subsections, the different concepts are analyzed in more detail.

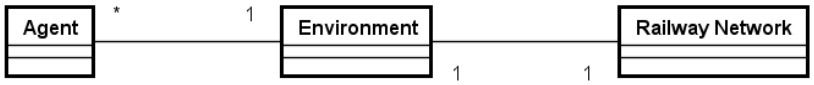


Figure 4: Domain model of Flatland Environment

3.2.1 Environment

The environment is the central concept for a simulation and holds the other two concepts, agent and railway network, together. This is also evident in the properties of the environment in Table 3, listing only the most important properties.

Table 3: Properties of environment

Property	Description	Values
Rail	Railway network	Railway network
Agents	All agents present in the environment	Array of agents
Max Steps	Maximum number of time steps before the simulation is aborted	Arbitrary integer (e.g., 1000)

Due to the introduction of malfunctions in Round 2, the environment has been enhanced with additional properties. The corresponding malfunction properties are given in Table 4. For the stochastic of malfunction occurrence, *Poisson* distribution is used as probability distribution. Each agent can suffer a malfunction at a random time. If this occurs, the agent cannot perform any action for a random period of time. This duration is limited by a minimum and a maximum of time steps.

Table 4: Malfunction properties of environment

Property	Description	Values
Rate	Poisson rate of malfunction occurrence of single agent	Arbitrary number (e.g., 12000)
Min Duration	Minimum duration of a malfunction	Arbitrary integer (e.g., 15)
Max Duration	Maximum duration of a malfunction	Arbitrary integer (e.g., 50)

3.2.2 Railway Network

3.2.2.1 Grid and Cells

A railway network consists of a two-dimensional rectangular grid with a defined number of cells in width and height. To distinguish between the different cells of the grid, a simple coordinate system is used. The rows and columns are numbered from the top left starting from zero. A cell is identified by its position in the form *(row, column)*. In Figure 5 the railway network from Figure 3 is covered by this coordinate system. For example, the cell at the bottom right-hand corner in Figure 5 is identified as (19, 34).

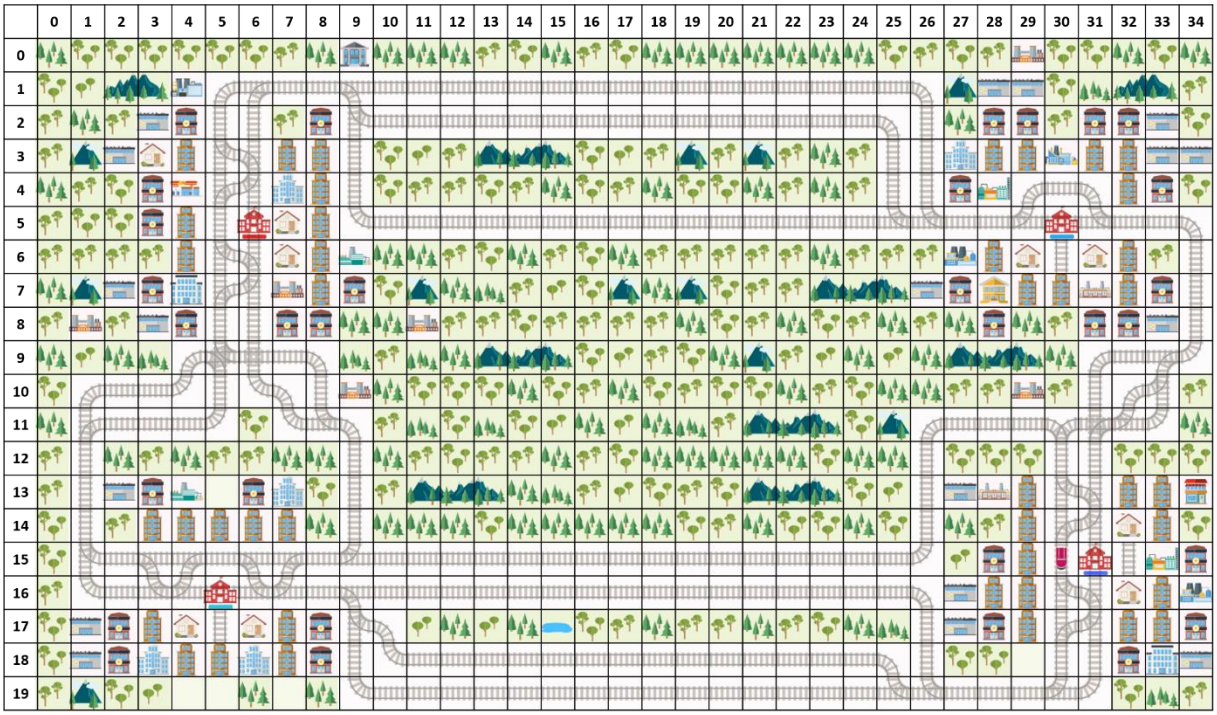


Figure 5: Railway network covered by the coordinate system

3.2.2.2 Transitions

A cell is characterized by so-called transitions, which an agent is allowed to perform on the cell or not. There are exactly 16 different transitions on a cell, which can either be allowed or forbidden. On the one hand, there are the four cardinal directions (north, east, south, west) in which an agent can be directed on the cell before the transition. On the other hand, the agent could also leave the cell in all four cardinal directions.

To keep this allow/deny decision for these 4x4 transitions, the Flatland Environment provides a bitmap of 16 bits for each cell, which is composed as follows:

Table 5: Structure of the transition bitmap

NN	NE	NS	NW	EN	EE	ES	EW	SN	SE	SS	SW	WN	WE	WS	WW

The first bit indicates whether an agent standing to the north on the cell can leave the cell to the north (NN). The second bit is used for the transition of a north-facing agent that wants to go east (NE). The other bits are assigned to the other transitions in the same way. Each transition is then either allowed (1) or forbidden (0). As an example, Figure 6 shows a cell which contains a switch and thus allows different transitions but forbids all others. Such a cell is also contained in Figure 5 at positions (1,9), (15,3), (15,6), and (16,9).

		Direction of exit			
		N	E	S	W
Original direction of agent	N	0	0	0	1
	E	0	1	1	0
	S	0	0	0	0
	W	0	0	0	1

Resulting bitmap: 0001011000000001

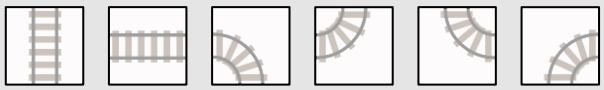
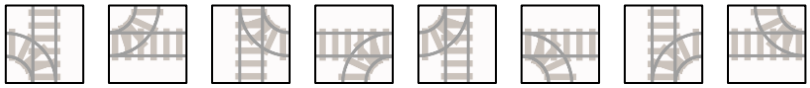
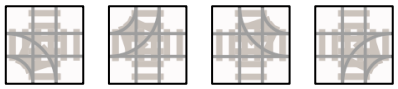
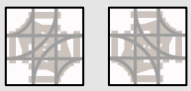
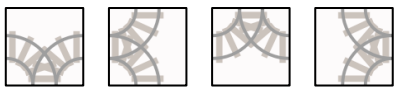
Figure 6: Example of a transition bitmap

3.2.2.3 Cell Types

In principle, any cell can be represented by means of a transition bitmap. However, not all possible combinations of transitions are valid in the Flatland Environment. This is due to the fact that junctions are realized by using a physical switch in the reality of a railway network. Thus, a maximum of two exit directions are allowed from any original orientation. This limitation results in eight different basic cell types, which are listed in Table 6. A cell can be rotated or mirrored to obtain another cell of the same cell type.

As can be seen in Figure 5, empty cells do not contain a track and are instead represented by trees, mountains or buildings for illustration purposes. If a cell is a target cell of at least one agent, it is decorated with a station building for illustration purposes. But in the background, it is still a normal cell with transitions. Such target cells are contained in Figure 5 at positions (5,6), (5,30), (16,5), and (15,30).

Table 6: Cell types

Cell type	Figures
Empty	
Straight	
Simple switch	
Diamond crossing	
Single slip switch	
Double slip switch	
Symmetrical switch	
Dead-end	

3.2.2.4 Properties

The present information of a railway network is stored very compactly as properties, which are summarized in Table 7.

Table 7: Properties of railway network

Property	Description	Values
Height	Height of the grid (number of rows)	Arbitrary integer
Width	Width of the grid (number of columns)	Arbitrary integer
Grid	Grid containing the transition bitmaps for all cells	2D array (height, width) of bitmaps

3.2.3 Agent

An agent in a Flatland Environment corresponds to a train on a railway network. In Figure 5, there is a single agent located in the cell (15,30), facing south.

3.2.3.1 Properties

An agent has a lot of meaningful information that is accessible through different properties. The most important properties of an agent are listed and described in Table 8.

Table 8: Properties of agent

Property	Description	Values
Handle	Index of the agent	0, 1, 2, 3, ...
Positions	• Initial Position: Start coordinates of the agent • Position: Current coordinates of the agent • Old Position: Previous coordinates of the agent	(row, column)
Directions	• Initial Direction: Start orientation of the agent • Direction: Current orientation of the agent • Old Direction: Previous orientation of the agent	0 (north) 1 (east) 2 (south) 3 (west)
Status	Current state of the agent	0 (ready to depart) 1 (active) 2 (done) 3 (done removed)
Target	Target coordinates for the agent	(row, column)
Moving	Current moving state of the agent	0 (not moving) 1 (moving)
Speed data	Speed profile of the agent (e.g., speed, position fraction)	

In Round 2 of the Flatland Challenge, different speeds and stochastic malfunctions are introduced. Accordingly, the properties of an agent are extended by a speed profile in Table 9 and a malfunction profile in Table 10.

Table 9: Speed properties of agent

Property	Description	Values
Speed	Speed of the agent	1/1 (fast passenger train) 1/2 (fast freight train) 1/3 (slow commuter train) 1/4 (slow freight train)
Position Fraction	Indicator, when the next action can be done according to the speed	Multiple of speed
Transition Action on Exit	Next action to be executed	(see Section 3.2.3.3)

The speed of an agent is fixed during the whole simulation. There is no acceleration or braking when starting or stopping. There are also no speed restrictions on specific cells. The agent speed results from the number of time steps the agent has to stay on the same cell before it can perform a next action. The speed property in Table 9 corresponds to the reciprocal of this number of time steps. In principle, other speed values would also be possible, but they must always be in the range between 0 and 1 (e.g., 1/5, 1/6, 1/7, etc.).

Table 10: Malfunction properties of agent

Property	Description	Values
Malfunction	Indicator, when the next action can be done according to the malfunction	0 (can move) 1 (wait 1 time step) 2 (wait 2 time steps) ...
Nr Malfunctions	Number of malfunctions that have already occurred on the agent	0, 1, 2, 3, ...

3.2.3.2 Agent Length

With the previous properties, the question arises where the agent length is defined. After all, trains are of different lengths in reality. This fact is ignored in the Flatland Environment. Instead, each agent has length 1 and thus occupies only one cell in the railway network at a time.

3.2.3.3 Actions

An agent can move from cell to cell through the grid but is restricted by the transitions in effect. The task and the goal of an agent are to move as quickly as possible from its start position to its target. For this purpose, there are different movement actions available for agents. Table 11 contains all actions that an agent can perform in the Flatland Environment.

Table 11: Agent actions

Action	Description
0: Do Nothing	If the agent is moving, it continues moving. If an agent is stopped, it stays stopped.
1: Move Left	If the agent is at a switch with an allowed transition to its left, it turns left. Otherwise, the action has no effect. If the agent is stopped, it starts moving (if allowed).
2: Move Forward	If the agent is stopped, it starts moving. If the agent is at a switch with an allowed transition straight ahead, it moves straight ahead. If the agent faces a dead-end, it turns 180 degrees.
3: Move Right	If the agent is at a switch with an allowed transition to its right, it turns right. Otherwise, the action has no effect. If the agent is stopped, it starts moving (if allowed).
4: Stop Moving	The agent stops at the current cell.

With the provided actions, it is noticeable that an agent cannot move backwards. The only exception is when an agent is on a dead-end cell. In this case, the train is turned around and can continue in the opposite direction.

3.2.4 Simulation Workflow

Having identified the most important components of the Flatland Environment, the question arises how the simulation is proceeded. The simulation of a Flatland Environment can be demonstrated as a workflow. This workflow is shown in Figure 7 and can be described as follows:

1. First, an environment containing the railway network and the agents is generated. There are several generators available in the Flatland Environment for this purpose. For example, environments can be loaded from a file or randomly built based on desired parameters.
2. Based on the current state of the environment and all agents, arbitrary observations from the view of the individual agents can then be built. For this purpose, all properties of the environment, of the railway network and of all agents can be accessed.
3. Based on these observations, the desired next action must then be selected for all agents.
4. After that, the selected actions of all agents are executed and the environment as well as all agents are updated accordingly.
5. If all agents have reached their target position in the railway network, the simulation is successfully completed.
6. If the defined maximum number of steps for the simulation is reached, the simulation is terminated prematurely. Otherwise, the simulation is continued with the next round at step 2.

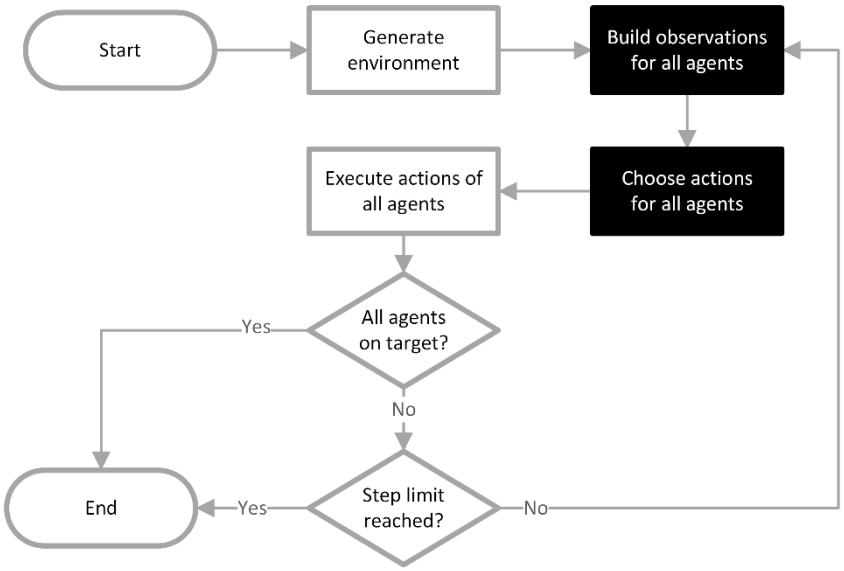


Figure 7: Simulation workflow

While the Flatland Environment takes over many tasks, two crucial tasks remain, which are colored black in Figure 7. The main task of the searched solution approach is to select the actions for all agents. The basis for this selection might be the observations, which can also be built by the solution approach. How such observations should look like and how the determination of the next actions should finally take place, is completely left to the solution approach.

3.3 Deadlock

Another part of the problem analysis concerns the possibility of so-called deadlocks. Deadlocks are widespread in computer science, especially in connection with processes in operating systems. According to (Coffman, Elphick, & Shoshani, 1971), the following conditions must be fulfilled simultaneously for a deadlock to occur:

1. **Mutual exclusion:** "Tasks claim exclusive control of the resources they require."
2. **Wait for:** "Tasks hold resources already allocated to them while waiting for additional resources."
3. **No preemption:** "Resources cannot be forcibly removed from the tasks holding them until the resources are used to completion."
4. **Circular wait:** "A circular chain of tasks exists, such that each task holds one or more resources that are being requested by the next task in the chain."

In the context of the Flatland Environment, the agents are the tasks and the individual cells in the railway network are the resources. Therefore, the deadlock conditions can be checked with regard to the Flatland Environment:

1. **Mutual exclusion:** A single cell in the railway network can be exclusively occupied by only one agent at a time.
2. **Wait for:** An agent always stands on a cell and thus has already allocated a resource. To move to the next cell, he has to allocate the next resource and wait for it if the next cell is occupied by another agent.
3. **No preemption:** An agent cannot be removed from the railway network until it has reached its target position. Thus, its allocated cell cannot be forcibly removed from it.
4. **Circular wait:** There is a chain of at least two agents, where each agent is on a cell that is the only next possible cell of the next agent in the chain.

The first three conditions are already fulfilled by the characteristics of the Flatland Environment and are therefore always met. Thus, the last condition remains as the only condition which can actually be influenced by the selection of actions for the agents.

Figure 8 contains some examples of railway networks with agents. In the first three examples, a deadlock has occurred. The crucial factor is that the agents cannot move backwards. For all agents, the possible next cells in forward direction are already occupied by other agents. Thus, no agent can move on and will never reach its target position. In contrast, examples 4 to 6 are similar but without a deadlock. There is at least one agent that can continue, and thereby freeing the cell for other agents.

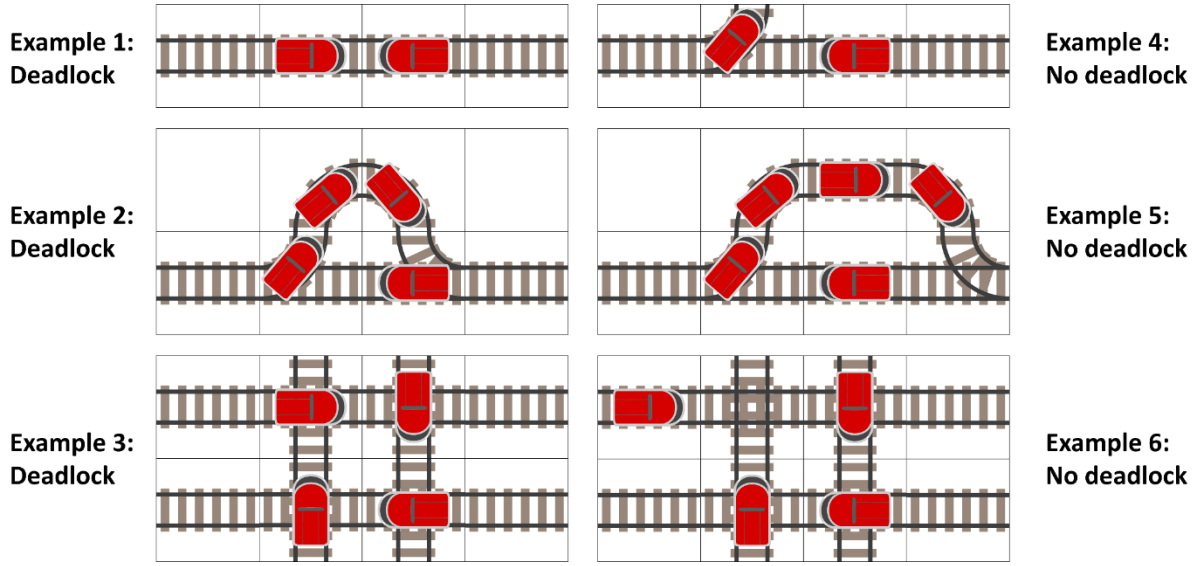


Figure 8: Railway networks with or without deadlock

In addition, there are situations which by definition are not yet a deadlock but will inevitably lead to a deadlock. Figure 9 contains some examples of such railway networks. In these situations, at least one agent can still move forward. But regardless of the next actions, a future deadlock with at least two agents cannot be prevented.

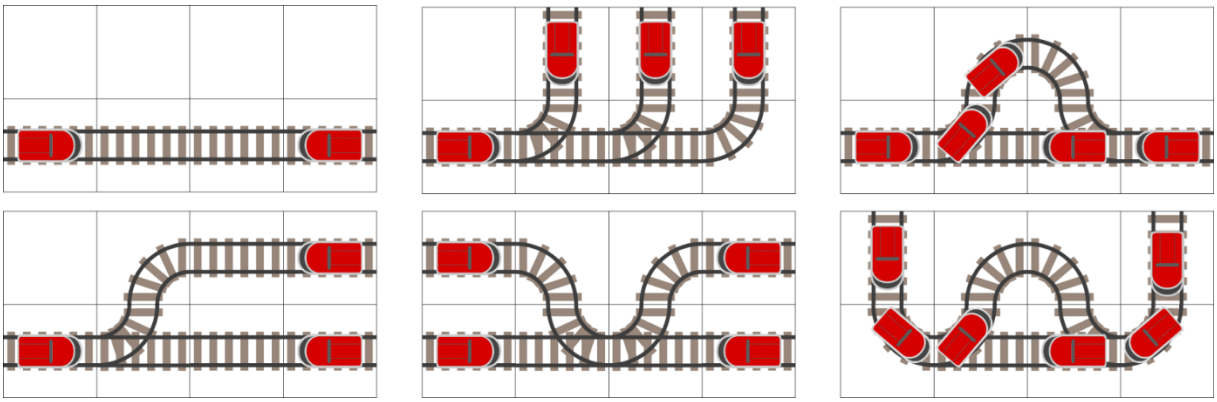


Figure 9: Railway networks with future deadlock

In conclusion, deadlocks are a very serious problem in the Flatland Environment. When a deadlock occurs, at least two agents are blocked. Deadlocks cannot be resolved because agents cannot move backwards in the Flatland Environment and no agents can be removed from the railway network. Thus, it is important to act with foresight and not to perform any actions that will later lead to a deadlock.

3.4 Conclusion

After the analysis of the problem, some aspects can be stated. In the previous sections, it was pointed out several times that some aspects of the reality are simplified very much or completely ignored in the Flatland Environment. In reality of railway networks, the following conditions apply, among others:

- Trains are of different lengths.
- Trains may also move in the opposite direction (backwards) under certain circumstances.
- Trains accelerate when starting and brake when stopping and do not always run at the same speed.
- There are different speed limits on track sections, which trains must adhere to.
- The route of trains is partly determined by any number of intermediate stations in addition to the starting station and target station.
- There is a fixed time schedule for trains, which must be followed.

Despite these strong simplifications in the Flatland Environment, the Flatland Challenge provides an excellent basis for researching approaches to the underlying problems MAPF and VRSP. However, it is questionable whether the findings from the Flatland Challenge can easily be transformed into the challenges in real railway traffic. Due to the given simplifications, the challenge is moving away from classical passenger traffic with timetables and mostly predetermined routes. Instead, it can be seen as a kind of traffic flow control, where different equivalent traffic units have to act without fixed routes and time planning.

4 Flatland Challenge: Round 0

Round 0 of the Flatland Challenge serves as an introduction and is intended to get familiar with the Flatland Environment. The task for this round can be described as follows according to Section 2.3.2:

> There is a railway network with a single train that has an assigned start position and an assigned target position. The train is located at its start position. The task of the train is to move on the shortest path through the railway network to its target position.

In other words, a solution approach must be developed that always selects as the next action the action that leads on the shortest path from the current position of the agent to its target position. Of course, only valid actions are allowed, namely actions for which a transition is permitted on the current cell.

4.1 Railway Network Encoding

As described in Section 3.2.2.2, the permitted transitions of a cell are stored in the form of a bitmap. This encoding is certainly very compact and has its advantages. Nevertheless, other encodings are also conceivable, which are more suitable for the determination of the shortest path in Round 0 but also for further rounds. In the following subsections, alternative encodings are explored. For all different encodings, the simple railway network in Figure 10 is always used as an example.

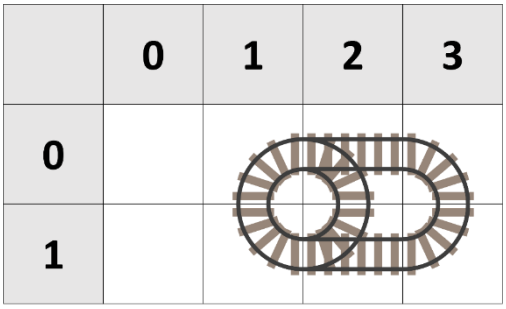


Figure 10: Railway network (example)

4.1.1 Cell Graph

A solution widely used in computer science is the encoding using a graph. A graph consists of any number of vertices and edges connecting the vertices. In order to represent a railway network as a graph, the individual cells can be interpreted as vertices and the allowed transitions between the cells as edges between the vertices.

Figure 11 contains a first attempt to represent the railway network of Figure 10 as a graph. As with the empty cells in the rail network, the corresponding vertices in the graph are unnecessary and could be omitted. The graph looks similar to the actual railway network. However, this representation as a graph is not correct, which is obvious from a simple example: In the real railway network, it is not allowed to move from cell (0,3) via cell (0,2) to cell (1,2). However, this would be possible in the resulting graph. This approach is therefore not valid for the encoding of a railway network.

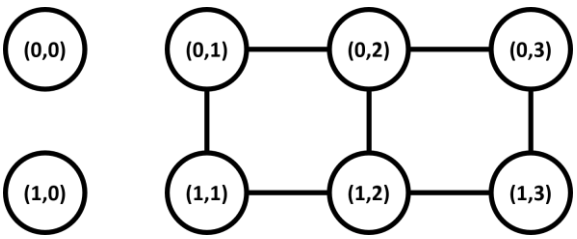


Figure 11: Cell graph of railway network in Figure 10

4.1.2 Cell Orientation Graph

To remedy the issue of the simple cell graph, the coordinate system is extended by an additional dimension, namely the orientation. In place of the vertex (0,0) for the cell (0,0), the four new vertices (0,0,0) for north, (0,0,1) for east, (0,0,2) for south and (0,0,3) for west are created. The identifier of a vertex is composed of *(row, column, orientation)*. Furthermore, directed edges are used. A directed edge from vertex (r1, c1, o1) to (r2, c2, o2) has the following meaning: If an agent is on cell (r1, c1) and is directed towards o1, it is allowed to move to cell (r2, c2) and will then be directed towards o2.

Figure 12 shows a cell orientation graph representing the railway network of Figure 10. For the sake of clarity, all vertices belonging to the same cell are marked gray together. In addition, all vertices that have no edges are omitted. Such a graph contains significantly more vertices than there are cells in the corresponding railway network, at most four times as many. In contrast to a cell graph, a cell orientation graph correctly represents a railway network.

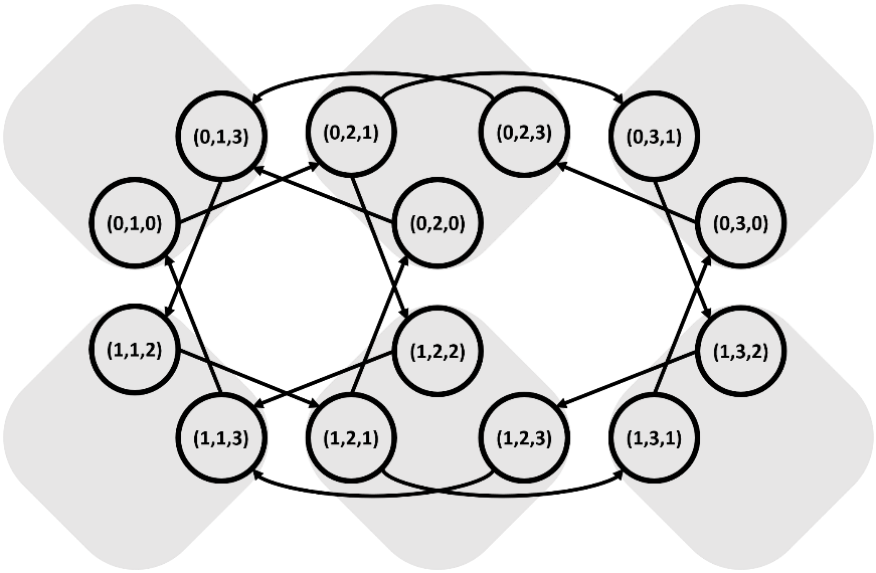


Figure 12: Cell orientation graph of railway network in Figure 10

4.1.3 Double Vertex Graph

A similar approach as the cell orientation graph is also followed with the so-called double vertex graph, as described in (Schlechte, 2012). Again, four vertices are created per cell. For the sake of simplicity, the same vertex identification *(row, column, orientation)* is used. However, the orientation in the identification no longer corresponds to the direction of the agent on the cell but to the direction of the exit. In addition, edges are not used to map the transitions between cells but the transitions within a cell in an undirected way.

Figure 13 shows a double vertex graph representing the railway network of Figure 10. For the sake of clarity, all vertices belonging to the same cell are marked gray together. In addition, all vertices that have no edges are omitted. The graph looks graphically the same as the railway network. Between two adjacent cells, there are two vertices that touch each other but have no edge in between. This means that a vertex has not only edges, but also exactly one partner vertex. This is the specialty of the double vertex graph and actually not compatible with a usual graph. Thus, the navigation through the graph must be adjusted. It is no longer allowed to go from one vertex via edge to the next vertex. Instead, it is only allowed to go from one vertex via the edge of its partner vertex to the next vertex. In other words, when navigating through the graph, two vertices must always be visited before the next edge can be used. In this sense, the path (0,3,1)(0,2,3) - (0,2,1)(0,1,3) is allowed, but the path (0,3,1)(0,2,3) - (0,2,1) - (0,2,0)(1,2,2) is not.

As with the cell orientation graph, this double vertex graph requires significantly more vertices than cells, but the railway network is represented correctly.

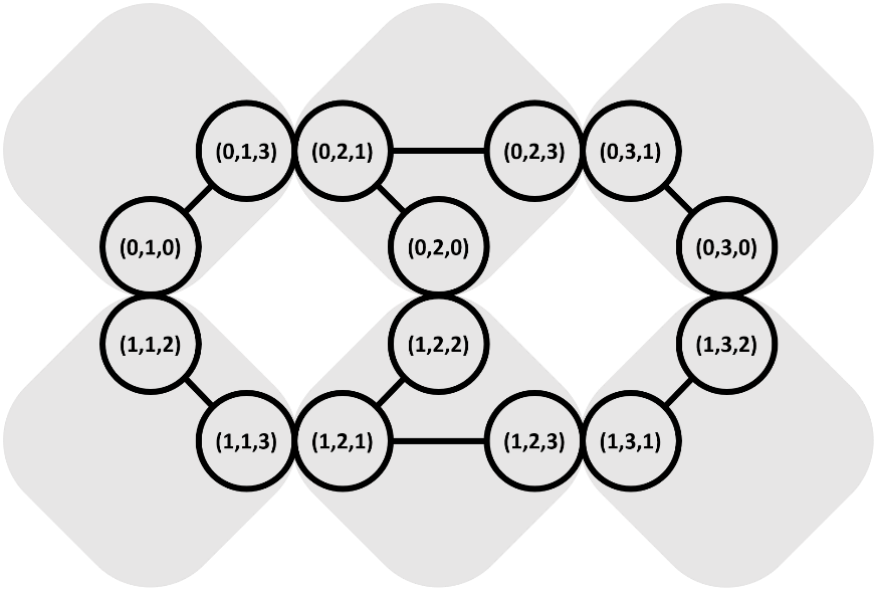


Figure 13: Double vertex graph of railway network in Figure 10

4.1.4 Implementation

Both the cell orientation graph and the double vertex graph can be used to encode the railway network. However, there are again different ways to implement the two graphs.

4.1.4.1 Adjacency List

A graph with vertices and edges can be implemented by hand with reasonable effort. A very trivial but effective variant is the form of an adjacency list. The first two columns of Table 12 show the adjacency list of the cell orientation graph in Figure 12, containing for each vertex a list of its adjacent successor vertices. If necessary and if helpful, the adjacency list can also be built for adjacent predecessor vertices (in the last column). In Python, a dictionary with a vertex as key and a list or set of vertices as value can be used for such adjacency lists. A double vertex graph can similarly be implemented with an adjacency list but must additionally be able to store the partner vertices.

Table 12: Adjacency list of cell orientation graph in Figure 12

Vertex	Successor vertices	Predecessor vertices
(0,1,0)	(0,2,1)	(1,1,3)
(0,1,3)	(1,1,2)	(0,2,0), (0,2,3)
(0,2,0)	(0,1,3)	(1,2,1)
(0,2,1)	(0,3,1), (1,2,2)	(0,1,0)
(0,2,3)	(0,1,3)	(0,3,0)
(0,3,0)	(0,2,3)	(1,3,1)
(0,3,1)	(1,3,2)	(0,2,1)
(1,1,2)	(1,2,1)	(0,1,3)
(1,1,3)	(0,1,0)	(1,2,2), (1,2,3)
(1,2,1)	(0,2,0), (1,3,1)	(1,1,2)
(1,2,2)	(1,1,3)	(0,2,1)
(1,2,3)	(1,1,3)	(1,3,2)
(1,3,1)	(0,3,0)	(1,2,1)
(1,3,2)	(1,2,3)	(0,3,1)

4.1.4.2 Library

Instead of implementing the graph itself, existing libraries can also be used. For example, the Python package *NetworkX* was used in the course of this study to implement the cell orientation graph. An advantage of such libraries is that they often provide additional functionality, such as for the calculation of the shortest path.

A double vertex graph is not a valid graph because of the partner vertices and cannot be modelled with a typical graph library. In order to use a library nevertheless, a corresponding wrapper can be implemented, which also provides the modified graph navigation.

4.2 Shortest Path Problem (SPP)

The original task of Round 0 of the Flatland Challenge requires to determine the shortest path for an agent from its start position to its target position in an arbitrary railway network. Since a railway network can be interpreted as a graph according to Section 4.1, the approaches of the well-researched graph theory can be used.

The so-called *Shortest Path Problem (SPP)* is one of the most well-known combinatorial optimization problems with various applications in theory and practice. The SPP can be described for graphs as follows according to (Korte & Vygen, 2018): Given is a graph with vertices and directed or undirected edges. All edges are assigned a weight. In addition, a start vertex and an end vertex are given. The task is to find the path between the start vertex and the end vertex so that the sum of the weights of all edges in this path is minimal.

For the weight of an edge, usually either the distance or the costs between both vertices are used. In railway networks and other applications of SPP where routes are involved, it is appropriate to use the distance between two vertices as the weight of the connecting edge. Since the railway network in the Flatland Environment is a regular grid and only neighboring cells can be connected by an edge, the distance 1 can be taken as weight for all edges in the graph.

4.3 Shortest Path Algorithms

There are several established algorithms to solve the SPP. The algorithms have different approaches and therefore are suitable for different scenarios. Table 13 summarizes the three most established algorithms and their purposes.

Table 13: Shortest path algorithms

Algorithm	Purpose
Dijkstra's Algorithm	Shortest path from a single start vertex to all other vertices in a graph with only non-negative weights
Bellman-Ford Algorithm	Shortest path from a single start vertex to all other vertices in a graph with positive and/or negative weights
Floyd-Warshall Algorithm	Shortest path from each vertex to each other vertex

With the first two algorithms, only the start vertex has to be specified, whereupon the shortest path to all other vertices and thus also to the target vertex is determined. In contrast, the third algorithm requires neither a start vertex nor a target vertex but calculates the shortest path between all possible vertex pairs. Since these algorithms are extremely widespread and well-known, no explanation of the exact procedures is given in this report. Usually, at least one of these algorithms is implemented in common software libraries for graphs. For example, *NetworkX* provides all three algorithms in different variations.[4]

[4] (Documentation: Algorithms: Shortest Paths, 2019)

4.3.1 Breadth-first Search (BFS)

In addition to the algorithms mentioned above, there are other ways to search a graph, such as the straightforward breadth-first search. This search is started at a start vertex to be specified. In the first step, all adjacent vertices of the start vertex are visited. In the next step, all adjacent vertices of all adjacent vertices of the start vertex are visited. This can be continued in the same way until all reachable vertices have been visited. If a vertex has already been visited in a previous step, it can be skipped.

On the first visit of a vertex, the path taken there always corresponds to the shortest path. However, this only applies because all edges have length 1. Otherwise, there could possibly be a shorter path. Since this condition is met in the Flatland Environment, this fact can be used to shorten the algorithm: Once the target vertex is visited, the shortest path is found and the algorithm can be terminated prematurely.

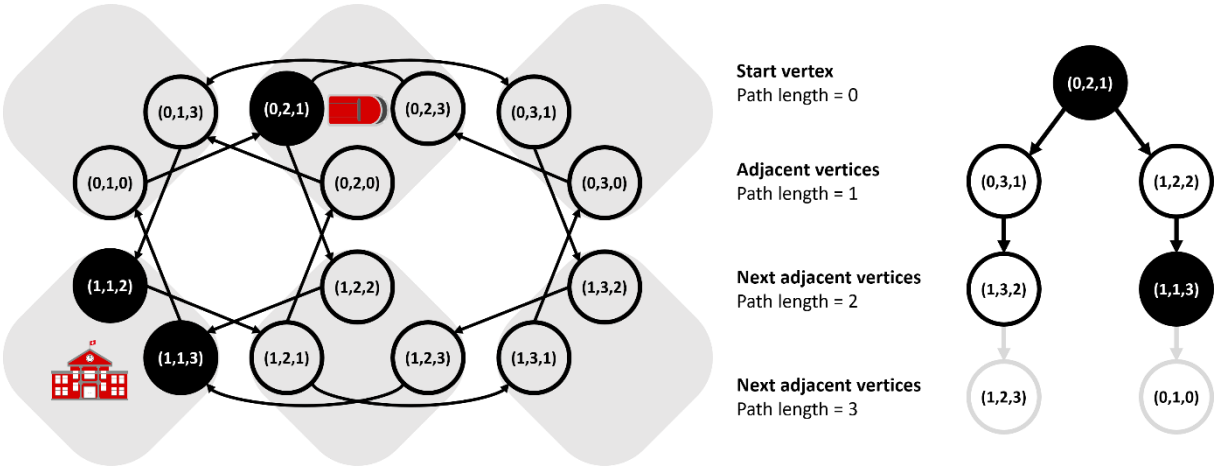


Figure 14: Breadth-first search in cell orientation graph

The procedure is demonstrated by the example in Figure 14. The cell orientation graph of Figure 12 is again shown on the left side. The agent is located on the cell (0,2) and is oriented to the east, which corresponds to the start vertex (0,2,1). Starting from this, the shortest path to the target cell (1,1) should be found using BFS. There are two feasible target vertices, namely (1,1,2) and (1,1,3), because it does not matter in which direction the agent is oriented on the target cell. In the first step, the adjacent vertices of the start vertex are visited. Within the same step, the vertices are visited in lexicographical order. Thus, the vertex (0,3,1) is visited first and then the vertex (1,2,2). The shortest path to these vertices has the length 1. In the second step, the adjacent vertices of (0,3,1) and (1,2,2) are visited, again in lexicographical order. By visiting the vertex (1,1,3) the target cell (1,1) is reached with the length 2 and the algorithm can be stopped. If the algorithm would be continued, two more vertices would be visited in the next step. However, the algorithm would then also be at the end, because all reachable vertices for the next step have already been visited.

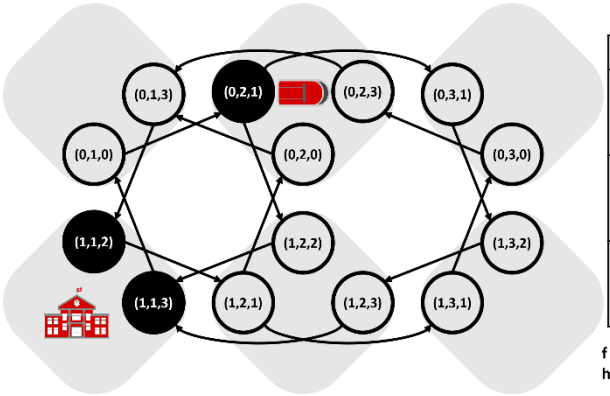
As has been shown, the BFS is well suited for finding the shortest path under the given circumstances. In fact, with a uniform edge length of 1, BFS corresponds to the procedure of the Dijkstra's algorithm.

4.3.2 A* Search

In an attempt to reduce the runtime of the graph search, another algorithm emerged. The core feature of the so-called A* search is that a heuristic is used to decide which vertex to visit next. The heuristic is used to estimate for each vertex the length of the remaining path to the target vertex. Thus, the next vertex to be visited is the one which leads to the shortest total length of the path, according to the heuristic.

Since the railway network in the Flatland Environment is a grid with coordinates, the *Manhattan distance* according to (Black, 2019) can be used as a simple heuristic. The Manhattan distance

between two cells is calculated as the sum of the absolute differences of the single coordinates. As an example, the Manhattan distance between the cells (0,3) and (1,2) becomes 2, because the absolute difference of the row coordinates ($|0 - 1| = 1$) and the absolute difference of the column coordinates ($|3 - 2| = 1$) are summed.



Iteration	Selected vertex	Open vertices	Closed vertices
1	(0,2,1)	(0,3,1) f = 1 h = 3 **f + h = 4**; (1,2,2) f = 1 h = 1 **f + h = 2**	
2	(1,2,2)	(0,3,1) f = 1 h = 3 **f + h = 4**; (1,1,3) f = 2 h = 0 **f + h = 2**	(0,2,1)
3	(1,1,3)	(0,3,1) f = 1 h = 3 **f + h = 4**; (0,1,0) f = 3 h = 1 **f + h = 4**	(0,2,1) (1,2,2)

f = path length from start vertex to current vertex
h = heuristic (Manhattan distance) from current vertex to target vertex

Figure 15: A search in cell orientation graph*

The procedure for the A* search is demonstrated in Figure 15 using the cell orientation graph from the example in Figure 12. Two sets are maintained during the search: The closed set contains all vertices that have already been visited. The open set contains all adjacent vertices of all closed vertices. In the first iteration, the start vertex (0,2,1) is selected as the next vertex and the adjacent vertices are inserted into the open set. Two values are determined for the vertices in the open set, namely the length of the shortest path from the start vertex (f) and the estimated remaining length to the target vertex (h). The heuristic used in the example is the Manhattan distance. In the second iteration, the selected vertex of the previous iteration is inserted into the closed set first. Then, the vertex promising the shortest total path (f+h) is selected from the open set. This vertex (1,2,2) is removed from the open set. Instead, the adjacent vertex (1,1,3) of the selected vertex is added to the open set. The iteration is repeated in the same way until the target vertex is selected. This is the case in the third iteration.

Note that the heuristic is of crucial importance. The efficiency of the search depends significantly on the heuristic. Even the Manhattan distance can provide incorrect estimates. Such a suboptimal estimation can be seen in the estimation for the vertex (0,1,0) in Figure 15: The Manhattan distance between the cell (0,1) and the target cell (1,1) returns 1, but the train would have to be able to travel backwards for this. Instead, the train must follow the small loop with length 3. However, with a good heuristic, the A* search is very helpful and is used by different approaches in later rounds.

4.4 Distance Map

It would be unnecessary and redundant if the shortest path and its length had to be determined repeatedly. It may therefore be useful to do this once at the beginning of the simulation. Thus, a so-called distance map was integrated into the Flatland Environment providing the minimum distance to the target cell for each agent on each cell with each orientation.

4.5 Conclusion

Round 0 of the Flatland Challenge is concerned with finding the shortest path. This problem is well-known as the shortest path problem and different algorithms exist to solve it. Both the elaborated encodings of a railway network and the shortest path algorithms will form the basis for the more difficult tasks in the next rounds.

5 Flatland Challenge: Round 1

While Round 0 of the Flatland Challenge was intended as an introduction, the problem to be solved is expanded for Round 1. The task for this round can be described as follows according to Section 2.3.2:

> There is a railway network with an arbitrary number of trains, that all have an assigned individual start position and an assigned individual target position. All trains are located at their start positions. The task is to move all trains through the railway network to their target positions. Thereby, the cumulated travel time of all trains should be as short as possible.

In other words, a solution approach must be developed that continuously selects the next actions for all agents so that all agents reach their target position and travel as short as possible in total. Of course, only valid actions are allowed, namely actions for which a transition is permitted on the current cell. If there is a conflict between several agents and thus a deadlock, the task cannot be completed.

An example of such a problem situation is given in Figure 16, which shows the railway network before the simulation starts. There are ten agents located at their start position. For each agent, a path has to be found which leads as fast as possible to its target position but without causing a conflict. Thus, the shortest path is not necessarily the optimal path anymore.

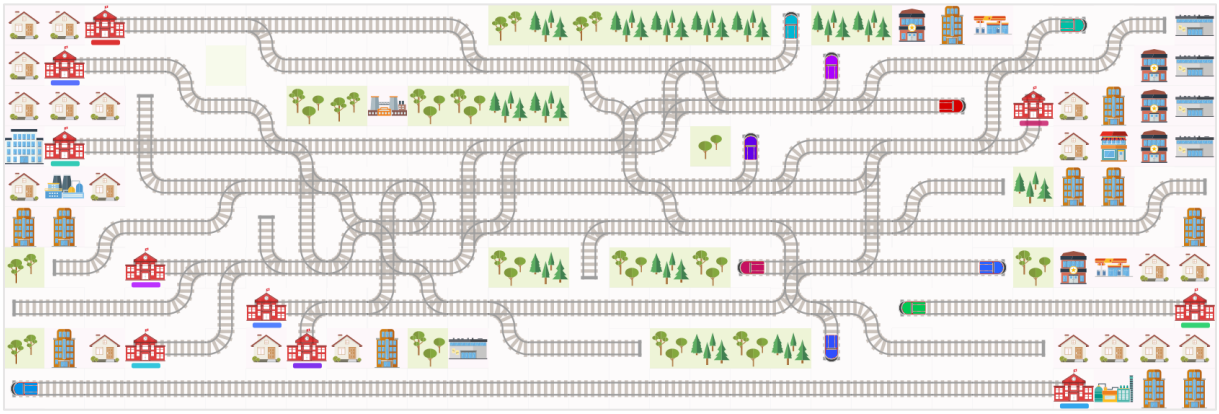


Figure 16: Railway network (Round 1)

5.1 Flatland Library

Since the underlying Python library was further developed during the Flatland Challenge, an older version must be used for Round 1. The compatible library is listed in Table 14.

Table 14: Flatland library for Round 1

Library	Version	Website
flatland-rl	0.3.10	https://pypi.org/project/flatland-rl/0.3.10

5.2 Test Cases

The organizers of the Flatland Challenge have defined 1,000 different test cases in the form of environments with a railway network and agents. These test cases are used to evaluate an algorithm and are kept secret. Instead, a public set of 20 similar test cases is provided. Table 15 lists these test cases and their properties. As a graphical example, test case 3.0 is shown in Figure 16.

The dimensions of the railway networks range from 10 rows and 10 columns to 100 rows and 100 columns. The number of agents in the environment ranges from 1 agent to 100 agents. In addition, there is a maximum number of time steps before the simulation of the environment is aborted. The right half of the table focuses on the number of cells in the environment and the

distinction into the different cell types (see Section 3.2.2.3). The cell type *Symmetrical switch* is included in the cell type *Simple switch* because both types are basically equivalent.

Table 15: Test cases of Round 1

Test case	Height	Width	Agents	Max steps	All cells	Track cells						
0.0	10	10	1	30	100	37	18	10	1	0	0	8
0.1	10	10	1	30	100	36	17	10	0	1	0	8
1.0	10	10	3	30	100	37	18	10	1	0	0	8
1.1	10	10	3	30	100	36	17	10	0	1	0	8
2.0	10	10	5	30	100	66	32	16	2	2	0	14
2.1	10	10	5	30	100	63	27	20	1	1	0	14
3.0	10	30	10	60	300	222	135	46	4	7	0	30
3.1	10	30	10	60	300	183	99	44	5	5	0	30
4.0	20	20	10	60	400	210	114	40	15	10	1	30
4.1	20	20	10	60	400	212	113	48	12	9	0	30
5.0	40	20	15	90	800	429	266	78	26	15	0	44
5.1	40	20	15	90	800	481	309	88	21	19	0	44
6.0	50	50	10	150	2500	694	554	92	14	4	0	30
6.1	50	50	10	150	2500	703	569	68	29	7	0	30
7.0	50	50	40	150	2500	1648	860	428	111	127	2	120
7.1	50	50	40	150	2500	1659	869	412	125	126	7	120
8.0	100	50	10	225	5000	1001	869	76	20	6	0	30
8.1	100	50	10	225	5000	1007	872	86	18	1	0	30
9.0	100	100	50	300	10000	4721	3199	926	280	159	7	150
9.1	100	100	50	300	10000	4686	3102	968	285	175	6	150

The various properties of the test cases allow an estimation of the complexity. The number of all cells (height * width) is not important, because extraneous empty cells are included as well. More useful is therefore the number of track cells, in other words non-empty cells. There are also differences in terms of track cells that can influence the complexity. The more transitions are allowed on a cell, the higher is the risk of conflicts between agents. But with more transitions, there can be more promising routes to the target position. Another important factor for the complexity of a test case is the number of agents. If there is only one agent, the railway network can be as complex as it is, but without really increasing the complexity.

5.3 Literature Research

To get an overview of the problem to be solved, a literature study was conducted. It turned out that the problem faced in this round of the Flatland Challenge has already been well-studied in various variations and is generally known as *Multi-Agent Path Finding (MAPF)*. A definition of this problem is provided by (Cohen, Wagner, Kumar, Choset, & Koenig, 2017):

> "Given an environment and agents with assigned start and goal locations, the Multi-Agent Path Finding (MAPF) problem is to find collision-free paths for all agents from their start to their goal locations that optimize some criterion such as makespan or sum of traveled distances."

The definition refers to *makespan* and *sum of traveled distances*. Makespan is the time span until the last of all agents has reached its target position. With the sum of traveled distances, the travel distance or the travel time of all agents from their start positions to their target positions is cumulated.

5.3.1 Fields of Application

An example for the application of MAPF is the management of a warehouse with autonomous driving units. Figure 17 shows such a warehouse in the form of a grid. The colored circles are agents. The black cells are the shelves and cannot be passed over. The white cells represent the corridors and can be passed over. All agents receive orders and have to drive to a certain shelf to store or pick material. Only one agent has room in the width of a corridor and conflicts should be avoided. In contrast to a railway network, a warehouse is usually very symmetrically structured, which could make the task easier.

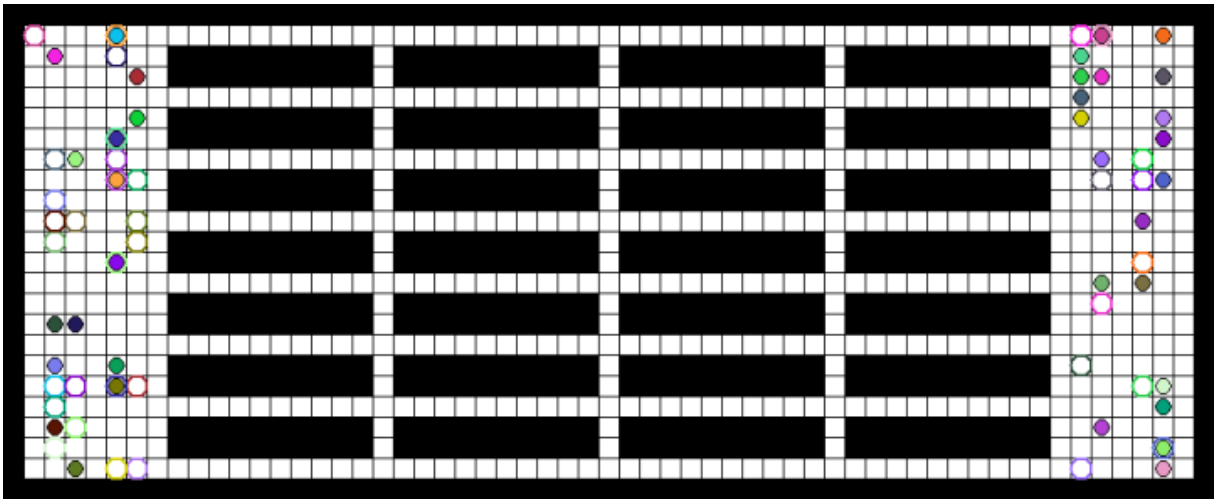


Figure 17: Warehouse grid (Ma, Kumar, & Koenig, 2016)

Another example for the application of MAPF is the classical *Sliding Tiles Puzzle* shown in Figure 18. There is a grid with 16 places and 15 numbered tiles. One place is empty. Compared to the number of agents, there is only one more place. The tiles next to the empty place can be moved to this place. The goal of the game is that finally all tiles are put in the right order from top left to bottom right. Only one agent can be moved at a time in each situation, which makes the problem simpler.

Figure 18: Sliding tiles puzzle, pictured by (Kelly, 2011)

5.3.2 Variations

There are also different variations of the problem with restrictive conditions or simplifying assumptions. In most cases in the literature, the MAPF problem is addressed using a simple grid, as shown in Figure 17 and Figure 18. An agent can move in the four cardinal directions if the

corresponding cell is not blocked by an obstacle or another agent. In the railway network of the Flatland Environment, the transitions of the cells determine in which directions an agent is allowed to move at all. Furthermore, diagonal movements are also allowed in some variations but not in the Flatland Environment.

When simulating a Flatland Environment, the agents are not moved simultaneously in one step but sequentially. In other variations this happens simultaneously and thus an agent can move to an already occupied cell if the occupying agent moves away at the same time.

There are also differences regarding the start positions and the target positions. For example, the target positions are not yet assigned at the beginning in some variations. In this case, it is necessary to determine which agent should ideally be assigned which target position. Another variation contains a central agent depot, which is the start position for all agents. It is then necessary to additionally determine the optimal order of the agents.

As the different examples and variations show, the problem is handled differently in the literature. There are several different approaches to solve the problem, with some approaches focusing only on a specific variation. Therefore, it is necessary to find out which algorithms can best be applied to the task at hand.

5.3.3 Complexity

The definition of MAPF can be divided into two tasks, namely finding a solution and optimality. According to (Andreychuk, Yakovlev, Atzmon, & Stern, 2019), finding a valid solution is feasible in polynomial time but finding the optimal solution is NP-hard. That means it is assumed that the problem cannot be solved in polynomial time, but it is not proven. In the following sections, approaches that provide optimal solutions are considered first, followed by approaches that provide suboptimal solutions but have a polynomial runtime.

5.4 Optimal Algorithms

As the literature study has shown, there are indeed approaches to optimally solve the MAPF problem. The following subsections describe the different optimal approaches and algorithms that have been implemented in the course of this study.

5.4.1 Linear Programming (LP)

As a first method, the mathematical approach is considered. For this purpose, an attempt is made to formulate a problem instance of the Flatland Environment as a mathematical optimization model.

5.4.1.1 Optimization Model

An optimization model consists of variables, parameters, constraints and an objective function and thus describes a problem in mathematical form. In order to find a solution for a test case, the properties of the corresponding environment are passed to the model in the form of parameters and then result in an instance of the optimization problem. These parameters determine the constraints and the objective function. It is then a purely mathematical task to find the corresponding variables that optimize the objective function in the desired manner (minimization or maximization) in compliance with the constraints.

5.4.1.2 Problem Formulation

For the formulation of the problem as a mathematical optimization model, the work of (Bartá̌k, Švancara, & Vlk, 2018) can be used as a basis. However, the approach to model formulation of the MAPF problem described therein must be adapted to the characteristics of the Flatland Environment.

A graph representing the grid of the railway network serves as a basis for the formulation. In summary, the formulation can be described as follows: The presence of an agent on a vertex is considered as an activity. Each activity has a start time and an end time, the difference of which is the length of the activity. An activity is optional, meaning it can be present or not. Three

optional activities are introduced per agent per vertex. The first activity corresponds to the time of the agent on the vertex. The other two activities describe the time of the agent on the incoming edge and on the outgoing edge of the vertex. There is also an optional activity per agent per edge. The path of an agent is finally defined by the presence of its optional activities. These activities can also be used to formulate the constraints to prevent the simultaneous presence of multiple agents on the same vertex and the crossing of agents on edges.

5.4.1.3 Solution Methods

Linear Programming (LP) is used to solve such an optimization instance. If all variables are constrained to integers, it is referred to as *Integer Linear Programming (ILP)*. If some variables are not discrete, it is referred to as *Mixed-integer Linear Programming (MILP)*. Over time, a variety of different so-called solvers have been developed for LP, ILP or MILP, which perform the solution of the mathematical system. To use this approach for the Flatland Environment, a test case has to be transformed into an optimization instance, so that this mathematical system can be solved by an existing solver.

5.4.1.4 Implementation

For the transformation of a test case into an optimization instance, two Python libraries were examined during the Flatland Challenge. Further information on the libraries used is given in Table 16. Both libraries enable the formulation of optimization problems on the one hand and the solving of problems with the help of included solvers on the other hand.

Table 16: Linear programming: Python libraries

Library	Version	Website
mip	1.3.15	https://python-mip.readthedocs.io/en/latest
PuLP	1.6.10	http://coin-or.github.io/pulp

5.4.2 Constraint-based Search (CBS)

However, besides the formulation as a mathematical problem, there are algorithmic approaches for the MAPF problem. With the so-called *Constraint-based Search (CBS)*, the new optimal algorithm for MAPF was first presented in (Sharon, Stern, Felner, & Sturtevant, 2015). However, the presented approach must be adapted to the characteristics of the Flatland Environment.

The CBS makes use of a so-called constraint tree, which is a binary tree with constraint nodes. A constraint node contains a solution, a set of constraints and the total cost of the solution (e.g., makespan or cumulated travel time). In summary, CBS works as follows:

First, an initial solution is produced consisting of the shortest paths for all agents. The root node of the constraint tree is built from this initial solution and an empty constraint set. This solution usually has conflicts between agents. From the root node, the first conflict between two agents (A1 and A2) is used to form two new child nodes. The first child node is given a constraint that prohibits agent A1 from performing the action that leads to the conflict. Then, a new shortest path is determined for agent A1, with respect to the constraint. This results in a new solution. If the solution is free of conflicts, the optimal solution is found. The same is done for the second child node and agent A2. The process described above is then repeated with the next node. For this, the node with the lowest solution costs is selected as the next node.

5.4.2.1 Example

The procedure is illustrated using an example shown in Figure 19. Three agents are located at their start positions and are to be moved to their target positions. The agents and their assigned target positions are marked with the same number and color.

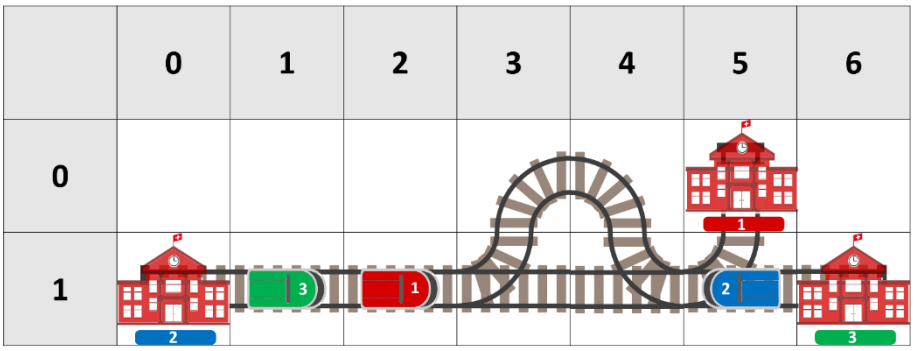


Figure 19: Railway network (example)

In a first step, the initial solution is determined consisting of the shortest paths for all agents. This results in the solution listed in Table 17 for the root node. For this solution, it is checked whether there are conflicts. The first conflict occurs between agent A1 and agent A2 in step 2: While agent A1 wants to move from cell (1,3) to cell (1,4), agent A2 wants to do the opposite. This is not allowed. Again in step 2, there is another conflict between A2 and A3, where both want to enter the same cell at the same time. However, only the first conflict is followed up. Note that there are no conflicts between agents A1 and A3, although A3 enters cells that have been occupied by A1 in the previous step. This is allowed because A1 is moved before A3 in each step due to the ascending order of agents.

Table 17: Initial solution for the root node

Agent	Step 0	Step 1	Step 2	Step 3	Step 4	Step 5
A1	(1,2)	(1,3)	(1,4)	(1,5)	(0,5)	*(0,5)*
A2	(1,5)	(1,4)	(1,3)	(1,2)	(1,1)	(1,0)
A3	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)

Two child nodes are derived from the first conflict between agents A1 and A2. The first child node is given the constraint *"A1 is not allowed to move into cell (1,4) in step 2"* and the second child node is given the constraint *"A2 is not allowed to move into cell (1,3) in step 2"*. For both child nodes, a new shortest path is then searched for the constrained agent, which complies with the given constraint. The corresponding solutions for the two child nodes are listed in Table 18 and Table 19.

Table 18: Solution for the first child node

Agent	Step 0	Step 1	Step 2	Step 3	Step 4	Step 5
A1	(1,2)	(1,3)	(1,3)	(1,4)	(1,5)	(0,5)
A2	(1,5)	(1,4)	(1,3)	(1,2)	(1,1)	(1,0)
A3	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)

Table 19: Solution for the second child node

Agent	Step 0	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6
A1	(1,2)	(1,3)	(1,4)	(1,5)	(0,5)	*(0,5)*	*(0,5)*
A2	(1,5)	(1,4)	(1,4)	(1,3)	(1,2)	(1,1)	(1,0)
A3	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	*(1,6)*

To decide which child node should be selected for the next round of the algorithm, the costs of the solutions are calculated. In this example, the cost function makespan is used. While in the first child node the last agent reaches its target in step 5, the last agent reaches its target in step 6 in the second child node. Thus, the first child node has the lower solution costs and is selected

as the next node. The solution of the selected node (in Table 18) is checked whether there are any conflicts. Again, the first conflict is between agent A1 and agent A2 in step 2, because both are located in cell (1,3). At the same time, even agent A3 is located in the same cell. However, only the first two conflict-causing agents are taken for a conflict. Based on this conflict, two new child nodes are created. The child nodes inherit the existing constraint *"A1 is not allowed to move into cell (1,4) in step 2"* from the parent node. In addition, one child node is given the constraint *"A1 is not allowed to move into cell (1,3) in step 2"* and the other child node the constraint *"A2 is not allowed to move into cell (1,3) in step 2"*.

The described process is repeated continuously until a node without conflicts is found. The solution of this node is optimal. For the sake of completeness, the optimal solution for the present example is listed in Table 20.

Table 20: Optimal solution

Agent	Step 0	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8
A1	(1,2)	(1,3)	(0,3)	(0,4)	(0,4)	(1,4)	(1,5)	(0,5)	*(0,5)*
A2	(1,5)	(1,4)	(1,4)	(1,4)	(1,3)	(1,2)	(1,1)	(1,0)	*(1,0)*
A3	(1,1)	(1,2)	(1,3)	(0,3)	(0,3)	(0,4)	(1,4)	(1,5)	(1,6)

5.4.2.2 Shortest constraint-fulfilling Path

During the execution of the CBS algorithm, the shortest path with respect to the given constraints must be determined for the corresponding agent on each node. For this purpose, the A* search (see Section 4.3.2) is very well suited. The A* search only needs to be extended by the constraint check: Before a vertex is visited, it is checked whether this visit would violate a constraint. If a constraint is violated, the vertex is rejected. In addition, a suitable heuristic is required for the A* search. The distance map (see Section 4.4) is well suited for this, because it provides the shortest real distance to the target position.

5.4.3 Operator Decomposition (OD) & Independence Detection (ID)

Another approach is a kind of cooperative path finding, consisting of the two components *Operator Decomposition (OD)* and *Independence Detection (ID)*. This approach presented by (Standley, 2010) also provides an optimal solution but had to be adapted to the Flatland Environment first.

5.4.3.1 Initial A* Approach

Using the A* search, the shortest path for a single agent can be determined as described in Section 4.3.2. In an obvious attempt, this algorithm can be extended to multiple agents. Briefly described, this can be realized as follows:

Instead of working directly with vertices, the extended algorithm works with nodes. This means that an open set of nodes and a closed set of nodes is maintained. A node contains the current positions of all agents. The initial node contains the start positions of all agents. From the initial node, all possible successor nodes are generated. To generate a successor node, a valid action must be taken for all agents. An action is valid if the corresponding cell transition is allowed. Another valid action is to wait on the same cell. For each generated node, it is checked whether the actions taken cause a conflict. If a conflict is found, the corresponding node is discarded. If not, the length of the remaining paths is estimated with the help of heuristics. Of all nodes that have not yet been processed (open set of nodes), the node that promises the best solution is then selected. Using the selected node, the same procedure is repeated until a node is found where all agents have reached their target position.

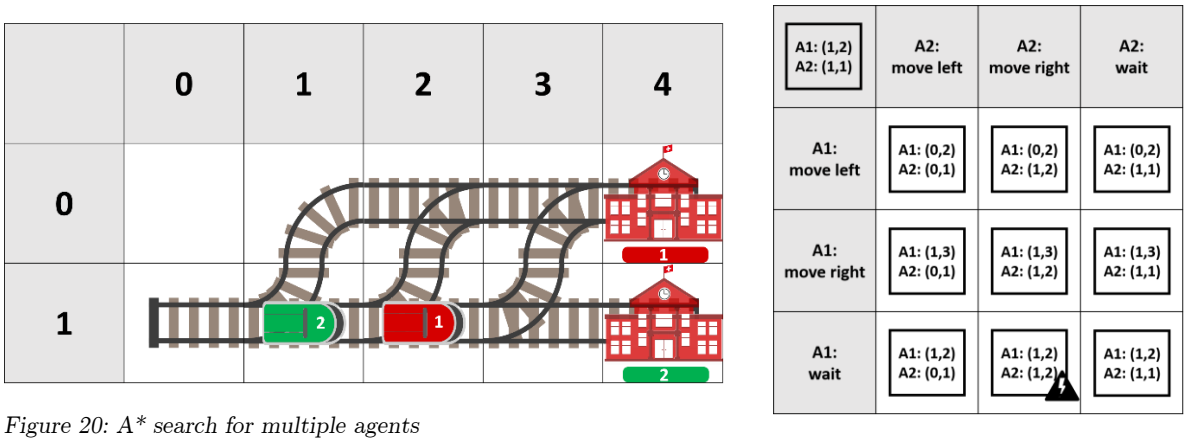


Figure 20: A search for multiple agents*

An example is presented in Figure 20 with a railway network and two agents. The initial node contains the positions (1,2) for agent A1 and (1,1) for agent A2. The matrix on the right side of the figure includes all possible successor nodes of the initial node. Only node [A1: (1,2) / A2: (1,2)] has a conflict and is discarded. All other nodes are added to the open set and are available to select as the next node.

According to Section 3.2.2.3, there are a maximum of two allowed transitions from the current position and orientation of an agent. In combination with the wait action, each agent can choose between a maximum of three valid actions in any step. Thus, there can be up to 3^n successor nodes for a node with n agents. This simple A* approach leads to an explosive increase of nodes.

5.4.3.2 Operator Decomposition (OD)

The MAPF can be solved with the initial A* approach. However, this approach can be improved by the so-called *Operator Decomposition (OD)*. Instead of the actions of all agents, only the action of the next agent in the sequence is considered. As a consequence, a single time step in the simulation no longer consists of only one node, but of n nodes. While the A* approach generates a maximum of 3^n successor nodes for all possible action combinations of the n agents, the OD approach only generates a maximum of three successor nodes corresponding to the valid actions of the next agent in the sequence.

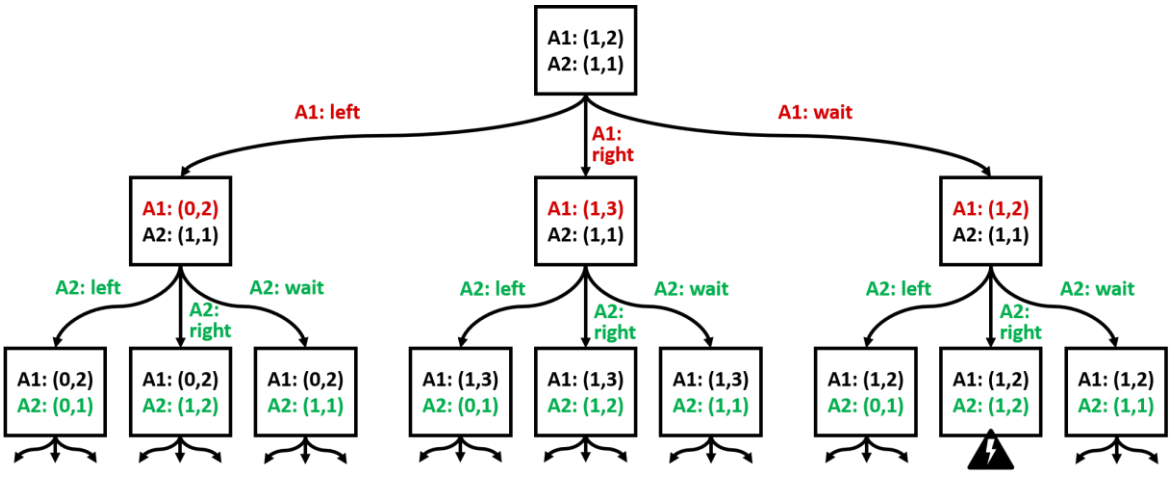


Figure 21: Operator decomposition (OD)

To demonstrate the difference to the A* approach, Figure 21 again shows the same example as in Figure 20 but using OD. From the initial node, only three successor nodes are derived, which correspond to the possible actions for agent A1. In the second level, three successor nodes would then be derived for each node again, which correspond to the actions of agent A2 resulting in the same nine nodes as with the A* approach. However, according to the principle of the algorithm, not all three nodes of the first level are continued with, but only the most promising one. In total, the decomposition of the actions reduces number of successor nodes (branching factor) and significant part of the search space does not have to be considered at all.

5.4.3.3 Independence Detection (ID)

The MAPF problem can be solved with the OD approach. However, instead of solving the entire problem with OD, the problem can be divided into smaller problems which are then solved with OD. The so-called *Independence Detection (ID)* was developed for this purpose. ID itself cannot solve MAPF, but is responsible for building the sub-problems, which are then solved by OD.

The ID procedure can be briefly described as follows: First, as many groups are created as there are agents. Each agent is assigned to a separate group. Then, the optimal solution is determined separately for each group. Since there is only one agent in each group, the optimal solution corresponds to the shortest path of the only agent. Afterwards, the solutions of all groups are checked against each other for conflicts. The two groups that cause the first conflict are considered for the next steps. For the first of these two conflicting groups, an attempt is made to find an alternative solution (using OD) that does not conflict with the other group and still has the same cost as the original solution. If this is not successful, the same is tried in reverse with the other group. If an alternative solution is found, this solution is taken for the corresponding group. If not, the two groups are merged and the solution for the new group is searched (using OD). Then, the solutions of all groups are checked again for conflicts. The described procedure is continued until no more conflicts occur. In this case, the optimal solution was found.

5.4.4 Timeout Issue

As shown in the previous sections, there are different algorithms that provide an optimal solution for the MAPF problem. However, all optional approaches do not have a polynomial runtime, as already mentioned in Section 5.3.3. This leads to an issue with the Flatland Challenge, because there is a time limit for the execution of the simulation. A maximum of 10 minutes is granted to calculate the solution of a test case before the simulation is aborted. This is no problem for the smaller test cases, but the larger test cases cannot be solved using the described algorithms within the time limit. For this reason, suboptimal approaches are considered as well.

5.5 Suboptimal Algorithms

While optimal algorithms always provide an optimal solution for the MAPF problem, this is not the case for suboptimal algorithms. Suboptimal algorithms also provide a valid solution that is as good as possible according to the algorithm, but mostly not optimal. Their greatest advantage is the complexity, since they can be executed in polynomial runtime. The following subsections describe the different suboptimal algorithms that have been considered in this study.

5.5.1 Optimal Anytime Algorithm (OAA)

The OD+ID algorithm described in Section 5.4.3 is optimal. In (Standley & Korf, 2011), an approach is presented that limits the runtime of this algorithm using a time limit. In this approach, the OD+ID algorithm is called repeatedly with increasing parameters and an ever-improving solution is found. Thus, the algorithm can be aborted anytime and then provides the best solution for that time. The name of the algorithm is misleading, because it is not the optimal solution, but the best solution so far at the time of abort.

The procedure can be described as follows: The OD+ID algorithm is extended by a new parameter for the maximum allowed group size (MGS). First, the OD+ID algorithm is executed with MGS = 1, whereby all agents are kept in their own group. This provides a solution very quickly, which is normally far from optimal. Then, the OD+ID algorithm is executed with MGS = 2, whereby a maximum of two agents may be combined in one group. With the help of lower bounds known from the previous execution, the algorithm execution can be shortened. This leads to an equally good or better solution. Subsequently, the MGS parameter is increased continuously and the OD+ID algorithm is executed again and again until the optimal solution is found, or the given time limit is reached.

5.5.2 Prioritized Planning (PP)

As an alternative approach, greedy algorithms must be considered as well. In the course of this study, a greedy algorithm for the MAPF problem was implemented, which is covered by the term *Prioritized Planning (PP)* in the literature[5].

The principle of PP is straightforward: First, all agents are put into a sequence according to certain criteria. Then, the first agent is considered according to this order and its shortest path is determined. In the sense of greedy, this path is final and will not be changed anymore. In the next step, the second agent according to the order is considered. For this agent, the shortest path is determined but without conflict with the previous agent. This path is again final and unchangeable. In the same way, the shortest conflict-free paths are determined for all remaining agents in the sequence.

5.5.2.1 Example

As an example, Figure 22 shows a very simple railway network containing two agents. With two agents, there are only two permutations and thus two possible sequences in which the agents can be handled. In the ascending planning order, agent A1 is processed first and then agent A2. In the descending planning sequence, it is in reverse. Both orders result in solutions, which are listed in Table 21 and Table 22. It can be seen that for this specific example, the ascending planning order provides a better solution with regard to both makespan and cumulated travel times.

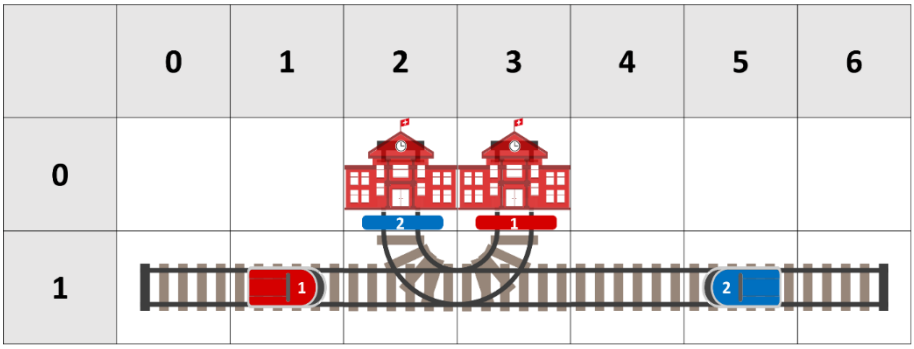


Figure 22: Railway network (example)

Table 21: Solution for ascending planning order (A1, A2)

Agent	Step 0	Step 1	Step 2	Step 3	Step 4	Step 5
A1	(1,1)	(1,2)	(1,3)	(0,3)	*(0,3)*	*(0,3)*
A2	(1,5)	(1,4)	(1,4)	(1,3)	(1,2)	(0,2)

Table 22: Solution for descending planning order (A2, A1)

Agent	Step 0	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7
A1	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)	(1,2)	(1,3)	(0,3)
A2	(1,5)	(1,4)	(1,3)	(1,2)	(0,2)	*(0,2)*	*(0,2)*	*(0,2)*

5.5.2.2 Prioritization

As the example shows, the solution is fully dependent on the chosen planning order. The correct prioritization of the agents is therefore of central importance. There are different ways to prioritize the agents based on their properties. For the Flatland Environment, different prioritizations can be used as described in Table 23. All described prioritizations can be applied in ascending or descending order.

[5] (Silver, 2005), (Cáp, Novák, Kleiner, & Selecký, 2014), (Ma, Harabor, Stuckey, Li, & Koenig, 2018)

Table 23: Agent prioritizations

Prioritization	Description
Identification number	The handle of each agent is considered (as in the previous example).
Shortest path length	The length of the shortest path of each agent is considered (using the distance map).
First conflict	The shortest path is determined for all agents. Then, it is considered for each agent at which time step the first conflict with another agent occurs.
Number of conflicts	The shortest path is determined for all agents. Then, the number of its conflicts is considered for each agent.

The list of prioritizations is not complete and other criteria could be explored. These prioritization criteria can either be used individually or combined as multi-level prioritization. For example, the agents can be sorted first by shortest path length, then by first conflict and finally by identification number. However, of all prioritizations, there is none that provides the best solution in all test cases or most of the test cases. Instead, it depends on the environment with the railway network and its agents which prioritization provides the best solution.

Fortunately, there is no need to decide on a prioritization at all. The algorithm can be executed separately for multiple prioritizations. This can be done either sequentially or even in parallel, because the executions are independent of each other.

5.5.2.3 Grouped Prioritization Planning

In the normal prioritization planning, one agent after the other is processed according to a defined order. Another approach extends PP to groups of agents. As with normal PP, the agents are put into a sequence. However, a group of next agents is then considered instead of the next agent individually. For this group, the best conflict-free partial solution is searched for and stored. Similar to the optimal anytime algorithm, the size of the group can be varied.

5.6 Conclusion

At the end of Round 1 of the Flatland Challenge, the following conclusion can be drawn: The fundamental problem of this round is known as Multi-Agent Path Finding (MAPF). There are several algorithms providing an optimal solution for this problem. However, these algorithms cannot be performed in a polynomial runtime. Since there is very little time available for calculations in real-time systems, suboptimal approaches are to be considered as well. Such algorithms also provide a solution for the MAPF problem, but this solution is not necessarily optimal. Thus, it is necessary to figure out which suboptimal approach provides the best results under the circumstances of the Flatland Environment. The corresponding results are discussed in Section 7.

The collection of approaches described in this section does not claim to be complete. There are other optimal and suboptimal approaches for MAPF, which are not discussed in this report.

6 Flatland Challenge: Round 2

Round 1 of the Flatland Challenge has revealed various solutions to the so-called Multi-Agent Path Finding (MAPF) problem. In the final Round 2, the problem to be solved is further extended and the highest difficulty of the Flatland Challenge is reached. The task for this round can be described as follows according to Section 2.3.2:

> There is a railway network with an arbitrary number of trains, that all have an assigned start position and an assigned target position. Multiple trains can have the same start position and/or target position. In addition, all trains have assigned a fixed and constant speed. At the beginning of the simulation, all trains are not yet on the railway network. With the first action of a train, it enters the railway network at its start position. Once a train has reached its target position, it is removed from the railway network again. The task is to move as many trains as possible to their target positions within the maximum allowed number of time steps. Thereby, the cumulated travel time of all trains should be as short as possible.

In other words, a solution approach must be developed that continuously selects the next actions for all agents so that all agents reach their target position and travel as short as possible in total. Of course, only valid actions are allowed, namely actions for which a transition is permitted on the current cell. If there is a conflict between several agents and thus a deadlock, the task cannot be completed.

For Round 2 of the Flatland Challenge, the generation of the railway networks was optimized by the organizer to get more realistic railway networks. In this way, separate stations with several parallel platforms can be simulated. As in reality, there are only a few track connections between the stations. An example of such a problem situation is given in Figure 23, which shows the railway network before the simulation starts. There are ten stations with up to four parallel platforms. Furthermore, no agents are visible, because they are not yet on the railway network at the beginning of the simulation.

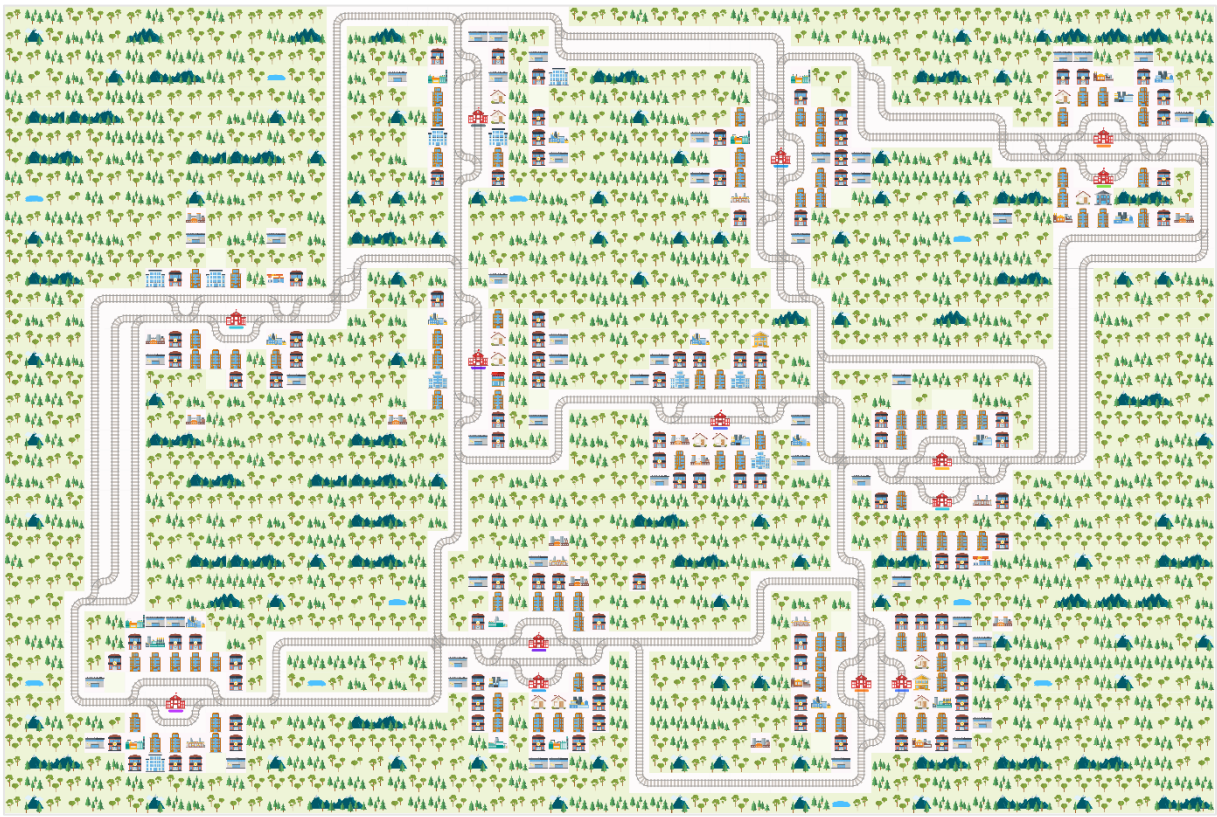


Figure 23: Railway network (Round 2)

6.1 Flatland Library

Since the underlying Python library is continuously developed further, a specific version must be used for Round 2. The compatible library is listed in Table 24.

Table 24: Flatland library for Round 2

Library	Version	Website
flatland-rl	2.1.10	https://pypi.org/project/flatland-rl/2.1.10

6.2 Test Cases

The organizers of the Flatland Challenge have defined 250 different test cases in the form of environments with a railway network, agents and malfunction occurrences. These test cases are used to evaluate an algorithm and are kept secret. Instead, a public set of 20 similar test cases is provided. Table 25 lists these test cases and their properties. As a graphical example, test case 4.1 is shown in Figure 23.

The dimensions of the railway networks range from 35 rows and 20 columns to 150 rows and 150 columns. The number of agents in the environment ranges from 50 agents to 200 agents. In addition, there is a maximum number of time steps before the simulation of the environment is aborted. The right half of the table focuses on the number of cells in the environment and the distinction into the different cell types (see Section 3.2.2.3). The cell type *Symmetrical switch* is included in the cell type *Simple switch* because both types are basically equivalent.

Table 25: Test cases of Round 2

Test case	Height	Width	Agents	Max steps	All cells	Track cells						
0.0	35	20	50	600	700	140	127	12	0	0	1	0
0.1	35	20	50	600	700	90	81	8	0	0	1	0
1.0	20	35	80	600	700	190	157	32	0	0	1	0
1.1	20	35	80	600	700	250	203	44	1	1	1	0
2.0	35	35	80	720	1225	212	166	44	0	1	1	0
2.1	35	35	80	720	1225	231	175	54	1	1	0	0
3.0	60	40	80	960	2400	374	277	90	4	2	1	0
3.1	60	40	80	960	2400	415	332	78	3	1	1	0
4.0	40	60	80	960	2400	399	319	76	0	0	4	0
4.1	40	60	80	960	2400	496	386	104	1	2	3	0
5.0	60	60	80	1120	3600	693	537	148	3	1	4	0
5.1	60	60	80	1120	3600	544	399	138	1	4	2	0
6.0	120	80	100	1760	9600	1346	1090	233	12	5	3	3
6.1	120	80	100	1760	9600	1203	988	198	6	3	6	2
7.0	80	100	100	1600	8000	1378	1048	316	6	4	4	0
7.1	80	100	100	1600	8000	1444	1121	316	4	1	2	0
8.0	100	100	200	1760	10000	1602	1242	346	7	1	6	0
8.1	100	100	200	1760	10000	1552	1166	376	1	4	5	0
9.0	150	150	200	2560	22500	2514	2097	390	10	9	8	0
9.1	150	150	200	2560	22500	2936	2492	420	5	9	10	0

As in Round 1, the various properties of the test cases allow an estimation of the complexity. Compared to the test cases in Round 1 (see Section 5.2), the dimensions of the test cases in Round 2 are considerably increased. Especially the number of agents is increased massively.

6.2.1 Malfunctions

As described in Section 3.2.1, the environment of each test case includes malfunction properties as well. These malfunction properties are identical for all present test cases and are listed in Table 26. Each agent can suffer a malfunction at a random time. This timing depends on the Poisson distribution using the specified rate 12,000. If a malfunction occurs, the affected agent cannot perform an action for a random number of time steps between 20 and 50.

Table 26: Malfunction properties of Round 2

Malfunction property	Value
Rate	12000
Min Duration	20
Max Duration	50

6.2.2 Maximum Number of Time Steps

According to the organizer of the Flatland Challenge, there is no guarantee that a test case can be solved completely within the maximum allowed number of steps. Nevertheless, as many agents as possible should be moved to their target position through the railway network within this step limit.

6.3 Problem Assessment

In a first step of Round 2, the present problem is analyzed how it has changed to the problem of Round 1. There are three main changes compared to Round 1, namely speed, malfunctions and start behavior.

6.3.1 Speed

While all agents have the same speed in Round 1, there are different speed profiles in Round 2. A speed profile determines how many time steps (typically 1, 2, 3 or 4) an agent must spend on a cell before it can perform the next action.

It was examined to what extent these different speeds affect the approaches from Round 1. It was found that this extension has no decisive influence on all approaches. All considered algorithms can be adapted and work for different speeds as well.

6.3.2 Start Behavior

In Round 2, the agents are not yet present on the railway network at the beginning of the simulation. Therefore, it is possible that several agents have the same start position. This circumstance extends the problem of Round 1 by another decision: It must be decided for all agents, when and in which order the railway network should be entered. Nevertheless, all considered algorithms from Round 1 can be adapted for this extension as well.

6.3.3 Malfunctions

While no malfunctions can occur in Round 1, this can happen in Round 2 at a random time for a random agent. These malfunctions introduce randomness to the problem. Since the stochastic malfunctions are not known in advance, they cannot be considered in the initial planning. Thus, the approaches from Round 1 are not readily usable in Round 2.

6.4 Literature Research

6.4.1 Problem Definition

To get an overview of the problem to be solved, a literature study was conducted. Thereby, one encounters the so-called *Vehicle Rescheduling Problem (VRSP)*, which was defined in (Li, Mirchandani, & Borenstein, 2007) as follows:

> "The vehicle rescheduling problem (VRSP) arises when a previously assigned trip is disrupted. A traffic accident, a medical emergency and a breakdown of a vehicle are examples of possible disruptions that demand the rescheduling of vehicle trips. The VRSP can be approached as a dynamic version of the classical vehicle scheduling problem (VSP) where assignments are generated dynamically."

This definition describes exactly the task to be solved. Henceforth, the term VRSP will be used for the problem given in Round 2.

6.4.2 Control Techniques

The fundamental problem is given by the task of the Flatland Challenge. However, there are two opposing techniques to address this problem.

6.4.2.1 Centralized Control

Nowadays, a traffic management system (TMS) is used at SBB and presumably at an overwhelming majority of all other traffic infrastructures. A TMS is a central unit that manages the planning for all agents collectively. All agents are thus subject to centralized control.

All approaches from Round 1 of the Flatland Challenge are classified to this control technique. Thereby, a solution for all agents is always determined centrally. Since optimal results are achieved with this technique, no other techniques were considered in Round 1.

6.4.2.2 Decentralized Control

In contrast to centralized control, there is the technique of dispensing with centralized coordination and instead handing over the planning to the individual agents in a decentralized manner. This decentralized control can be referred to as *Vehicle Based Train Control (VBTC)* system. In this sense, each agent is independently responsible for selecting its next action.

6.5 Rescheduling

A first approach to solve the VRSP is already contained in the name of the problem: rescheduling. It is an attempt to apply the algorithms used for the MAPF problem to the VRSP as well.

The principle is very simple: First, an initial and optimal solution for all agents is determined using a MAPF algorithm. To do this, the corresponding algorithm has to be adapted to the different speed profiles and the changed start behavior. Then, the simulation is started with all the agents following their determined paths. This continues until a malfunction occurs. Since the original solution is no longer feasible due to the malfunction, a new solution must be determined from the current positions of the agents. This can again be achieved with an adapted MAPF algorithm. The described process is repeated continuously until all agents have reached their target positions or the time limit is exceeded.

6.5.1 Deadlock Issue

This approach is intuitive and seems promising. However, it was found that the described approach cannot work completely with the characteristics of the Flatland Environment. This is because an occurring malfunction can lead to a deadlock. The railway network in Figure 24 is used as an example to demonstrate this issue.

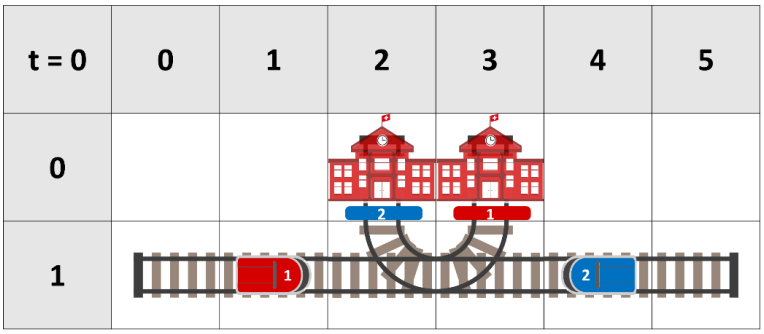


Figure 24: Rescheduling simulation (t=0)

According to the approach, an initial solution is determined for the given example using a MAPF algorithm. In the example, agent A1 has speed 1/1 and agent A2 speed 1/3. Thus, A1 needs only one time step for an action, but A2 needs three time steps. The corresponding solution is listed in Table 27.

Table 27: Initial solution

Agent	Speed	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9
A1	1/1	(1,1)	(1,2)	(1,3)	(0,3)	*(0,3)*	*(0,3)*	*(0,3)*	*(0,3)*	*(0,3)*	*(0,3)*
A2	1/3	(1,4)	(1,4)	(1,4)	(1,3)	(1,3)	(1,3)	(1,2)	(1,2)	(1,2)	(0,2)

The simulation of the example is started. Agent A1 chooses to move forward as the first action so that it will be on cell (1,2) at time t=1. Agent A2 also chooses to move forward as the first action. The action of A2 takes three time steps, so that he will be on cell (1,3) at time t=3. The situation of the simulation at time t=1 is shown in Figure 25.

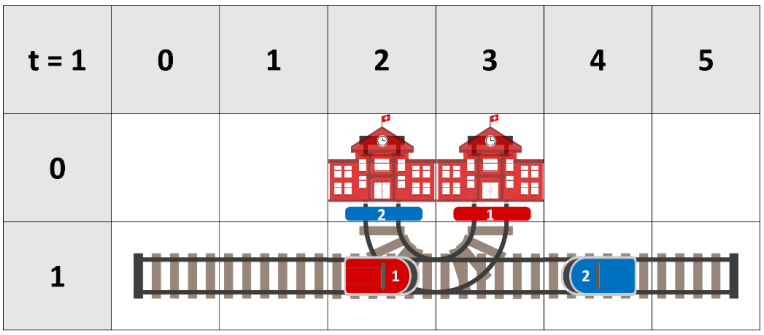


Figure 25: Rescheduling simulation (t=1)

In the next step, agent A1 chooses to move forward as the next action to get to cell (1,3). Agent A2 cannot yet choose a next action because it is still waiting for its previously chosen action to be executed. Now, a malfunction occurs on agent A1. Instead of moving to cell (1,3), A1 has to stay on cell (1,2) in a defective state. The situation of the simulation at time t=2 is shown in Figure 26.

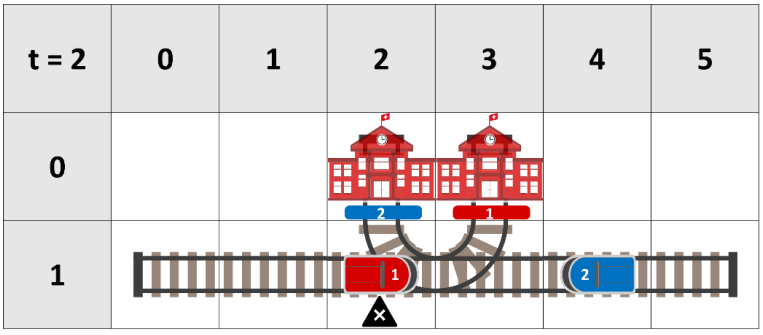


Figure 26: Rescheduling simulation (t=2)

In the next step, agent A1 cannot choose an action because of the malfunction. Instead, the action for agent A2 is finally executed and it moves to cell (1,3). This results in the situation shown in Figure 27 causing a deadlock, where both agents are blocked.

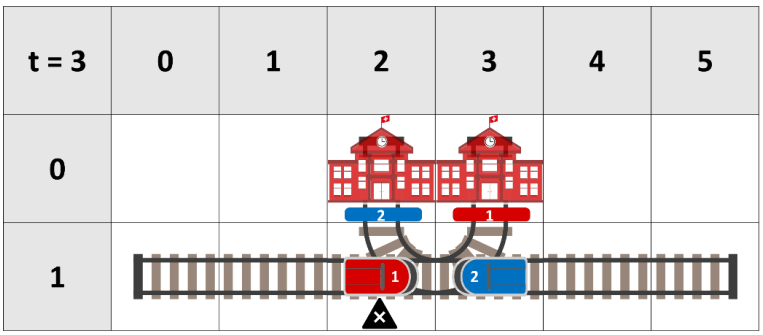


Figure 27: Rescheduling simulation (t=3)

6.5.2 Conclusion

The preceding example showed how quickly a malfunction can lead to a deadlock. Under these circumstances, rescheduling using existing MAPF algorithms is not appropriate. For this reason, the rescheduling approach is not considered further.

6.6 Waiting

A similar and even simpler variant as rescheduling is the waiting approach. In this variant, no new solution is determined when a malfunction occurs. Instead, all agents without malfunction are paused as well and it is waited until the malfunction is over. When the malfunction is over, the original plan is continued at the same place.

However, this approach has some drawbacks. On the one hand, up to 50 time steps are simply lost by waiting for each malfunction. In the worst case, a new malfunction occurs again and again within 50 steps, and all agents are just waiting until the time limit expires. On the other hand, there is the same deadlock issue as with rescheduling. For these reasons, the waiting approach is not considered further as well.

6.7 Complete Path Reservation (CPR)

The approaches discussed so far are susceptible to deadlocks caused by malfunctions. Therefore, an approach is wanted which can eliminate this circumstance as far as possible. This type of planning is defined as *Robust Scheduling* in (Zuo, 2009):

> "For an uncertain scheduling problem, the goal of robust scheduling is to generate a suboptimum scheduling scheme that is not sensitive to stochastic disturbances, i.e., robust scheduling emphasizes on the stability of scheduling schemes."

Thus, an approach is to be developed that cannot lead to a deadlock in the event of a malfunction. In the course of this thesis, such an approach was elaborated and given the name *Complete Path Reservation (CPR)*.

6.7.1 Control Technique

This approach differs from previously discussed approaches in the control technique: While in the other approaches all paths for all agents are calculated in advance using a central control technique, this is not the case in the present approach. Instead, each agent decides anew in each step which action it wants to perform next. This procedure corresponds to the decentralized control technique described in Section 6.4.2.2.

6.7.2 Reservation

The concept of reservation is introduced as the basis for this approach. Each track cell in the grid of the railway network can be reserved by agents in the different directions. Thereby, a cell can be reserved by multiple agents in the same direction. However, it is not allowed to reserve a cell in one direction if a reservation in the opposite direction already exists for this cell.

6.7.3 Procedure

To describe the procedure of this approach, the example environment shown in Figure 28 is used. The railway network consists of two stations with three platforms each and a total of three agents. The agents are not yet present in the railway network at the beginning of the simulation. However, the start positions and target positions of the agents are marked in the figure: Agent A1 is to be moved from cell (0,3) to cell (0,9), agent A2 from cell (1,3) to cell (1,9) and agent A3 from (2,3) to (2,9).

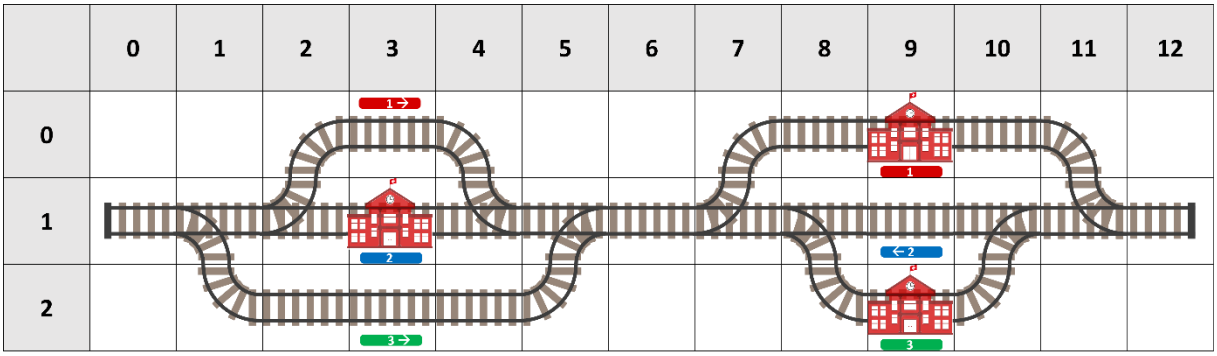


Figure 28: CPR simulation (initial state)

The simulation is started with the first time step. Agent A1 has to select its first action. A1 tries to find the shortest path from its start position to its target position, for which all cells are not yet reserved in the opposite direction. Since no reservations have been made yet, A1 finds the path (0,3) - (0,4) - (1,4) - (1,5) - (1,6) - (1,7) - (0,7) - (0,8) - (0,9). In consequence, A1 reserves all cells of this path in the corresponding directions and enters the railway network. The reservations are marked in Figure 29.

Next, agent A2 tries the same procedure. However, A2 cannot find a complete path where all cells are not yet reserved in the opposite direction. Cells (1,7), (1,6), (1,5) and (1,4) are already reserved from A1 in the opposite direction. Therefore, A2 is not yet allowed to enter the railway network and does not perform any action.

Next, agent A3 tries the same procedure and finds the path (2,3) - (2,4) - (2,5) - (1,5) - (1,6) - (1,7) - (1,8) - (2,8) - (2,9). Cells (1,5), (1,6) and (1,7) are already reserved by A1, but in the same direction. In consequence, A3 reserves all cells of this path in the corresponding directions and enters the railway network. The reservations are marked in Figure 29.

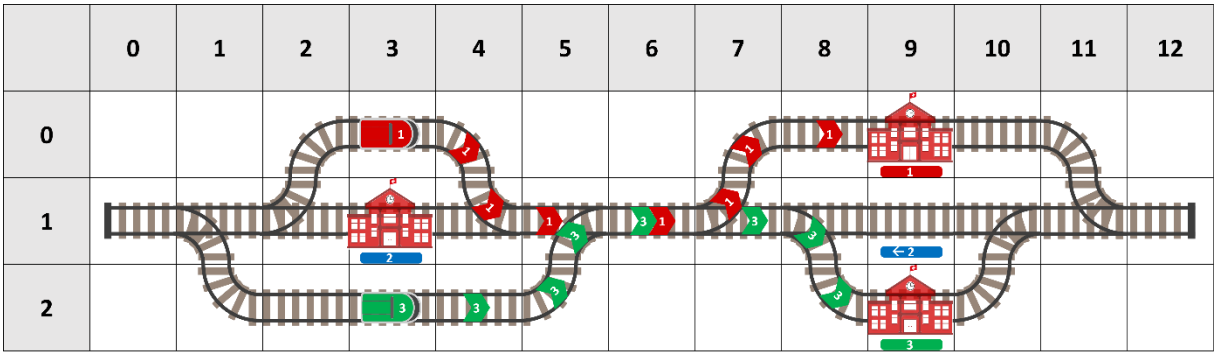


Figure 29: CPR simulation (step 1)

The simulation is continued with the second time step. Agent A1 has already reserved a complete path to its target position. Nevertheless, A1 now checks if there is a shorter, non-reserved path in the meantime. This is not the case and A1 continues to follow its reserved path. When an agent leaves a cell of its reserved path, the reservation for that cell is removed. Then, agent A2 tries to find a complete non-reserved path again but fails again. As agent A1, agent A3 cannot find a shorter non-reserved path and continues to follow the already reserved path.

The same procedure is repeated at each time step. For example, Figure 30 shows the situation after the fifth time step. The agents A1 and A3 are moving on their reserved paths, while agent A2 is still not allowed to enter the railway network.

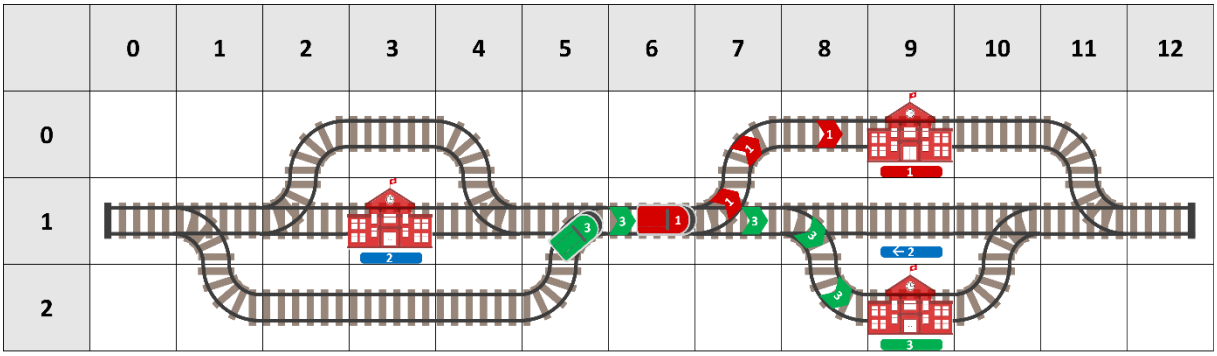


Figure 30: CPR simulation (step 5)

This situation only changes four time steps later, when agent A3 releases the reservation of cell (1,8). As shown in Figure 31, agent A2 can then finally reserve a complete path and enters the railway network.

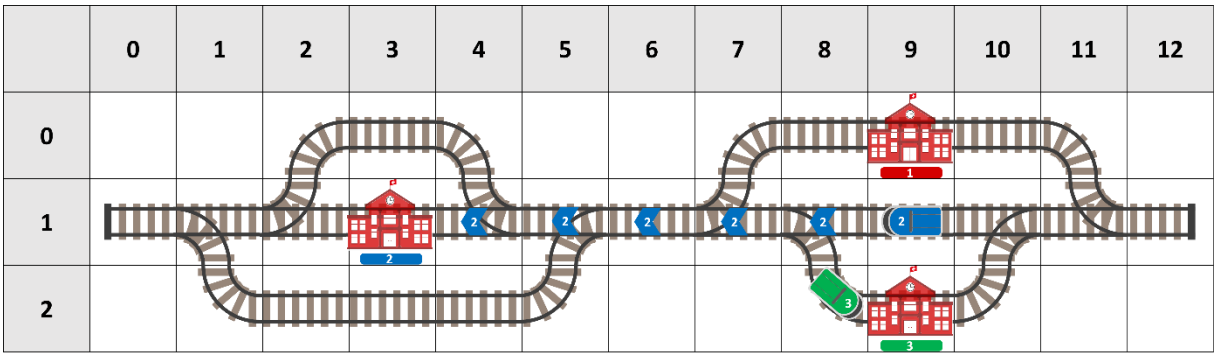


Figure 31: CPR simulation (step 9)

6.7.4 Agent Prioritization

As described in the procedure, each agent acts independently and tries to reserve a complete path. Nevertheless, there is an order in which the agents can do this. In the preceding example, the default prioritization with ascending agent numbering is used. Instead, a different prioritization could be used, where agent A2 is first to act. In this case, A2 would have reserved a complete path first and the simulation would have been different.

Thus, it is evident that the prioritization of the agents is of great importance. As already described in Section 5.5.2.2, there are many different ways to prioritize the agents. Table 28 lists only some of the possible prioritization criteria for agents. In addition, there are also criteria for agents that have not yet entered the railway network, listed in Table 29. All described criteria can be applied in ascending or descending order.

Table 28: Prioritization criteria for agents

Prioritization criterion	Description
Identification number	Agents with a lower number are prioritized.
Status	Agents which are moving in the railway network are prioritized over agents which have not yet entered the railway network.
Speed	Agents with a higher speed are prioritized.
Shortest path	Agents which a lower minimum distance from the current position to the target position (according to the distance map) are prioritized.
Shortest non-reserved path	For all agents, the currently shortest non-reserved path from the current position to the target position is determined. Agents with a shorter such path are prioritized.
Number of steps	For all agents, the expected number of time steps to the target position is determined depending on the length of the shortest path. Agents with a lower number of steps are prioritized. Instead of the shortest path, the shortest non-reserved path can be considered as well.

Table 29: Prioritization criteria for non-started agents

Prioritization criterion	Description
Number of agents (same position)	Often, there are multiple agents with the same start position. Agents with many other such agents are prioritized.
Number of agents (same position & orientation)	Often, there are multiple agents with the same start position and the same start orientation. Agents with many other such agents are prioritized.
Average speed (same position)	For each start position, the average speed of the other agents is calculated. Agents with a higher such speed are prioritized. Instead of the average, the minimum or maximum can be considered as well.
Average speed (same position & orientation)	For each start position and orientation, the average speed of the other agents is calculated. Agents with a higher such speed are prioritized. Instead of the average, the minimum or maximum can be considered as well.

Although the above criteria can be used individually, they are intended for combined use as multi-level prioritization. For example, the agents can be sorted first by status, then by speed and finally by the length of the shortest path. Ten different criteria have been described, all of which can also be used in reverse. With these criteria, the prioritization can have up to ten levels. Thus, there is a countless number of different combinations of criteria. However, it depends on the environment with the railway network and its agents which prioritization provides the best solution. Therefore, the combination should be chosen which leads to the best results on average.

6.8 Reinforcement Learning

Due to the development in the field of artificial intelligence in recent years, various novel methods have emerged in computer science. A promising approach from the field of machine learning is the so-called *Reinforcement Learning (RL)*.

6.8.1 Introduction

The basic idea of reinforcement learning is that a software agent autonomously learns what it should ideally do in which situation. For this purpose, the learning behavior in nature is simulated.

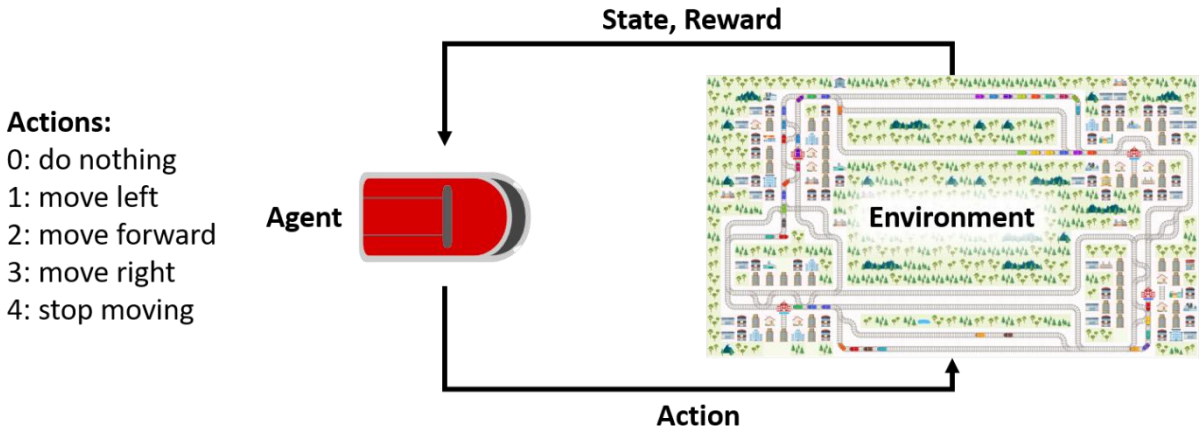


Figure 32: Reinforcement learning: scheme of the learning cycle

The functionality of RL is introduced using the scheme in Figure 32. The two main components at RL are the agent and the environment. Other important concepts are action, state and reward. With the help of RL, an agent should acquire the knowledge about which action should be chosen in which state in order to maximize the reward. In the Flatland Environment, the agent is known to be a single train. Furthermore, the environment is known as a railway network including all agents. An agent can perform five different actions, as described in Section 3.2.3.3.

The learning takes place in a cycle between the agent and the environment: The agent receives a state, which describes the current situation of the environment. Based on this state, the agent should decide which action to take. Since the agent has not yet acquired any knowledge, it randomly selects one of the possible actions and executes it in the environment. By executing the action, the environment changes its state. Therefore, the new state of the environment is again passed on to the agent. In addition, the agent receives a reward from the environment as well. The reward is a value that indicates whether the previous action was good or not. The higher the reward, the better the action was. Thus, the agent learns which action has led to which reward in which state. This learning cycle can now be continued for any period of time and the knowledge of the agent can be deepened.

The principle of RL so far is simple. Nevertheless, there are some concepts which need to be explained in more detail.

6.8.1.1 Reward

In the form of a reward, the agent receives a feedback from the environment about how good the action was. As a practical example, the dressage of animals can be mentioned: If an animal performs a stunt successfully, it is given a treat. In contrast, no treat is given if the animal refuses the stunt.

With the Flatland Environment, the matter is a little more extensive. If the agent performs a single action, it is not immediately known how good it is. In order to find out the actual worth of an action, future rewards must also be considered. Only when the agent reaches its target position with an action, it is immediately clear that this action was very good.

For the weighting of the future rewards, a so-called *Discount Factor* is used in RL. This factor is between 0 and 1 and determines to what extent future rewards are included. As an example, a factor of 0.8 means that the current reward is weighted with 1, the first future reward is weighted with 0.8^1, the second future reward with 0.8^2 and so on. It must be experimented which value for the discount factor leads to the best result.

6.8.1.2 Selection of Action

As described, the agent randomly selects its action at the beginning of the learning process because it does not yet have any knowledge. However, knowledge is built up with each further cycle run. This then leads to the dilemma between exploration and exploitation: Should the agent continue to randomly select the action in order to continue exploring new actions that may also lead to success or should it select the action based on its knowledge (exploitation)?

In RL, it has become established to start with exploration and then slowly but continuously move towards exploitation. For this purpose, a factor called *Epsilon* is introduced. Epsilon represents the probability of exploration. At the beginning of learning, epsilon is equal to 1, so the action is chosen randomly. Then, its value is continuously reduced until it is almost 0. For example, a value of 0.6 means that the action is chosen randomly with a probability of 60% or decided based on the existing knowledge with a probability of 40%.

However, the question is how much epsilon should decrease after each cycle. As with the discount factor, experiments must be carried out to find out which kind of decrease leads to the best result.

6.8.1.3 State

The current situation of the environment is passed on to the agent in the form of a state. The state corresponds to the observation referred to in section 3.2.4. A state usually consists of a collection of numerical values representing the situation in the environment. However, it is completely open how an observation is to be made from the current situation of the environment and how a state is to be derived from it.

6.8.1.4 Knowledge

The agent learns and builds up knowledge, but how is this done? The method used for this is called *Deep Q-Learning*. This kind of learning is carried out with the help of a *Neural Network*, which consists of several layers. The scheme of such a neural network is shown in Figure 33.

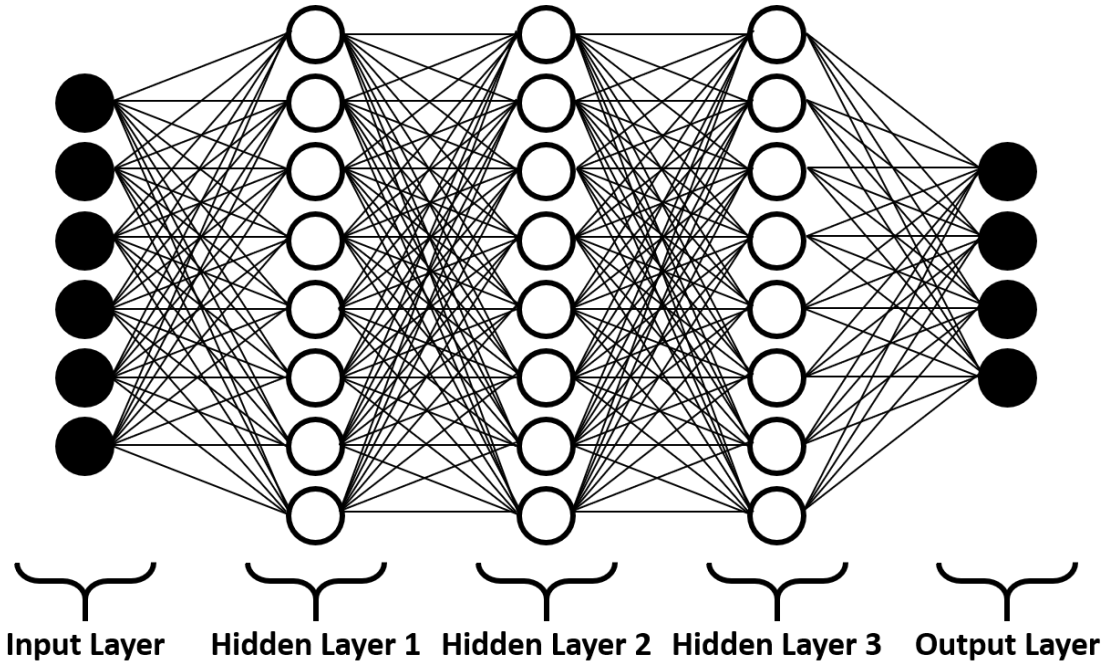


Figure 33: Deep Neural Network

The first layer is the input layer that corresponds to the state. The last layer is the output layer, which contains a value as quality (Q) for each possible action. The action with the highest Q-value corresponds to the best action for the current state in the input layer. A function has to be specified which converts the state from the input layer into the qualities of the actions in the output layer. The parameters of this function are optimized with each learning cycle. To achieve a better result, any number of so-called hidden layers can be used as intermediate layers between the input layer and the output layer.

6.8.1.5 Library

For programming in connection with RL, it is recommended to use an existing software library. For the research in the course of this thesis, the Python library *PyTorch* was used. PyTorch was developed by a Facebook research team and can be used for building neural networks to model the learning process.

6.8.2 Challenges

After the introduction to RL, the question arises how the theory of RL can be applied in order to acquire the knowledge necessary to solve the fundamental problem of VRSP. This task is related to different challenges. The field of machine learning and especially of RL is very novel and little researched. So far, this approach is mainly used for various games for demonstration purposes. In addition, there are many degrees of freedom in RL due to the different parameters which significantly influence the learning success. Therefore, it requires a lot of effort and experiments to coordinate all components of RL to each other.

6.8.3 Multi-Agent Reinforcement Learning

As stated in the objective of the fundamental task, the cumulative travel time of all agents should be minimal. Thus, the main focus is not on the actions of a single agent, but on the cooperation of all agents. Ideally, the agents should take each other into account instead of concentrating on themselves.

A first approach to achieve this is to extend the normal RL approach to *Multi-Agent RL*. For this purpose, the scheme is slightly adapted, as shown in Figure 34. The current situation of the environment is handed over to the agents in the form of a join state, consisting of separate partial states for the individual agents. The next action is then selected for each agent and all actions are transferred combined to the environment as a joint action. This results in a single reward that judges the quality of all actions as a whole.

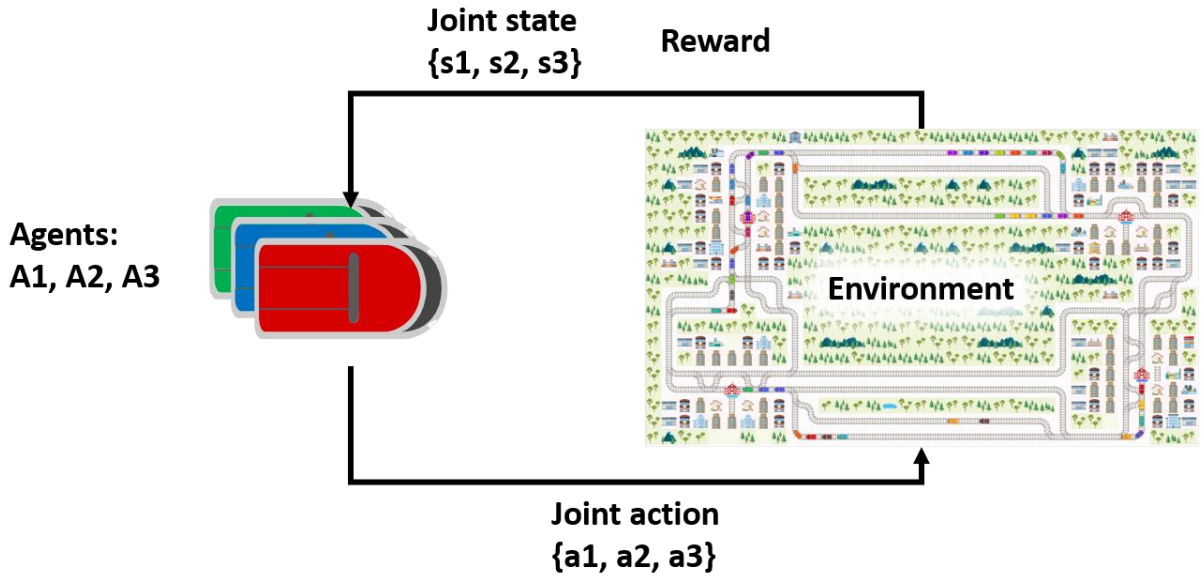


Figure 34: Multi-agent reinforcement learning

In the given context of the Flatland Environment, there are some difficulties in realizing this approach. The main issue is the variable number of agents. It is not known in advance how many agents are present in a railway network. In order for the RL model to build up its knowledge, it is important that the number of agents is always constant. For example, if a multi-agent RL model is trained with five agents, it cannot be applied to ten agents afterwards. For this reason, the multi-agent RL approach is not pursued further and instead the focus is on the standard (single agent) RL approach.

6.8.4 RL Approach

In the course of this thesis, the focus is therefore on the standard RL approach with only one learning agent. This approach is to be adapted to the circumstances of the Flatland Environment. The following subsections describe how the individual components are applied to the Flatland Environment.

6.8.4.1 Reward

In the Flatland Environment, the agent should reach its target position as soon as possible. In order to achieve this, the number of actions performed is to be minimized. For this purpose, a negative reward is returned with each action. Only if the agent arrives at its target position by executing an action, a positive reward is returned for this action.

In order to determine the exact reward values that lead to the best learning progress, it is necessary to experiment again. For the attempts in this thesis, the following values are used: If an agent reaches its target position with an action, the agent receives a reward in the amount of the maximum number of allowed time steps of the environment. In all other cases, a negative reward of -1 is returned.

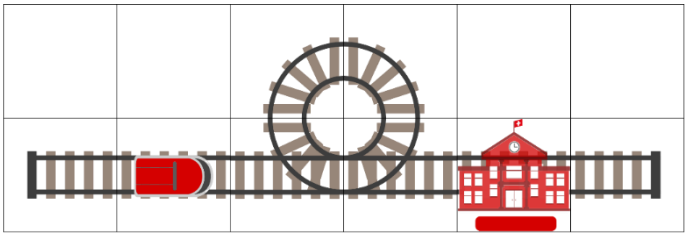


Figure 35: Railway network with simple loop

The effects of this specification can be demonstrated using the simple example railway network in Figure 35. It is assumed that the railway network has a maximum of 10 allowed steps. The agent has speed 1 and can therefore perform a maximum of ten actions before the simulation is aborted. If the agent drives directly to its target position, three actions are used. For the first two actions, the agent receives a reward of -1 and for the third action a reward of +10. In the second case, the agent will move into a loop first. Thus, it first receives a reward of -1 six times and then a reward of +10. In the third case, the agent passes the loop twice, which causes the simulation to be aborted before the agent has reached its target. In this case, the agent receives a negative reward of -1 ten times and never a positive reward. Taken all the rewards together, it results in a total reward of +8 in the first case, +4 in the second case and -10 in the third case.

6.8.4.2 Actions

As stated in Section 3.2.3.3, there are five different actions for an agent. Accordingly, five values are calculated with the help of a neural network, which are to indicate the quality of the respective actions. It is obvious that the choice between many actions is more difficult to make than with fewer actions. Therefore, it is advisable to keep the number of available actions as small as possible.

One of the five possible actions is the *Do Nothing* action. This action differs from the other actions as it does not give a direct movement instruction but is used to repeat the previous action. However, instead of selecting the *Do Nothing* action, the previous action can be selected again with equal effect. Therefore, the action is redundant and will be omitted henceforth to minimize the number of possible actions.

Furthermore, it is evident that not every action makes sense in every situation. For example, an agent cannot perform the *Move Left* action at all if there is no track connection to the left. In the context of RL, such impossible actions are referred to as *invalid actions*. To optimize learning, an agent is prevented from selecting an action that is invalid in the current state.

6.8.4.3 State

As in general in RL, there are many different possibilities how the state could look like in the context of the Flatland Environment. In the course of this thesis, two different approaches to build the state are considered: global observation and local observation. These two approaches are introduced and described in Section 6.8.5 and Section 6.8.6

6.8.4.4 Knowledge

The learning is done with the help of a neural network, illustrated in Figure 36. The observed state is used directly as input for the neural network and the output corresponds directly to the qualities of the four allowed actions. The qualities are always calculated for all actions including the invalid actions. The invalid actions are only ignored when searching for the action with the maximum quality. This also applies if the agent selects an action at random.

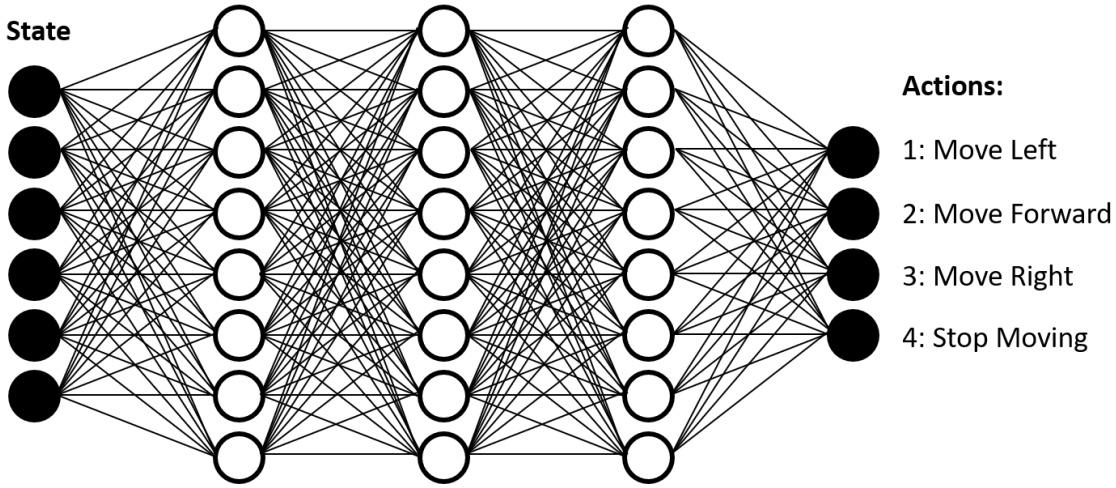


Figure 36: Neural Network with state and actions

6.8.5 Global Observation

As already mentioned, RL is currently mainly used for playing computer games. The most commonly used and therefore classical approach is simple: The image of the screen of the game is used as the state of the environment. Based on this image, the agent should then learn, for example, where it is currently located and what it should do next.

6.8.5.1 Classic State

The state of the environment is thus given by an image. This image must be transformed into a numerical state for the input layer of the neural network. Fortunately, images can be represented using the RGB color space. RGB consists of three channels for the colors red (R), green (G) and blue (B). Every single pixel of the image is represented by the proportion of red, green and blue.

The entire image is stored in a three-dimensional array. The first dimension corresponds to the three channels (RGB) and the other two dimensions are the height and width of the image in pixels. As an example, an image with a height of 100 pixels and a width of 200 pixels results in an array with the dimension (3, 100, 200). In total, this amounts to 60,000 individual values, which are the input for the neural network.

6.8.5.2 Adapted State

This classical approach can also be used for the Flatland Environment. However, a first problem already appears before the first step of the simulation: On the image, all agents are missing because they have not yet entered the railway network. Therefore, the image alone obviously does not provide all the information needed.

For this reason, new structures are defined based on the classical idea. Instead of the individual pixels, the individual cells in the grid of the railway network are considered. In addition, the classical channels (RGB) are replaced by new channels. The channels should represent different properties of the respective cells, describing the current situation of a cell. Thus, the resulting array has the dimension *(number of cell properties, height of the grid, width of the grid)*.

With the help of meaningful channels, the current situation of a cell should be expressed. In the course of this thesis, 44 different channels have been defined for this approach. The channels can be divided into three groups. The first group includes the general properties of the corresponding cell, as shown in Table 30. This concerns the transitions that uniquely identify the type of cell. Furthermore, it is recorded how many agents have the respective cell as start position or target position.

Table 30: Channels and properties (cell)

Channel	Property	
1 - 16	**Transitions**	The first 16 channels correspond to the transition bitmap of the cell (see Section 3.2.2.2). Each of the 16 possible transitions is either allowed (1) or forbidden (0).
17 - 20	**Number of agents before start**	These four channels indicate how many agents have the cell as the start position and have not yet entered the railway network. The four channels correspond to the four cardinal points in which an agent can be oriented at the start.
21	**Number of agents with target**	This channel records how many agents have the cell as the target position, excluding those that are already done.

As already mentioned, the state of an environment corresponds to the observation for a specific agent. Therefore, it should also be recorded on which cell the respective agent is currently located. The properties described in Table 31 are intended for this purpose. All important properties of the agent are included. It is to be noted that these channels only hold the properties at the cell where the observed agent is currently located. An exception is channel 33, where the property is only present at the target cell of the agent. For all other cells, these channels are empty.

Table 31: Channels and properties (observed agent)

Channel	Property	
22	**Handle**	Identification number of the observed agent
23	**Direction**	Orientation in which the observed agent is currently directed
24	**Moving**	Indicator whether the observed agent is currently moving or not
25	**Speed**	Speed of the observed agent
26	**Position fraction**	Position fraction of the observed agent
27	**Malfunction**	Number of steps until the malfunction of the observed agent ends
28	**Nr malfunctions**	Number of malfunctions that have occurred on the observed agent so far
29 - 32	**Distance to target**	Minimum distance from the current cell to the target of the observed agent, divided into the four possible orientations (north, east, south, west)
33	**Target**	Indicator whether this cell is the target position of the observed agent

In addition to the observed agent, it is also necessary to know about all other agents in the railway network. Therefore, the channels used for the observed agent are duplicated to be used for the other agents. If another agent is located on a cell in the railway network, the properties of this agent are stored in the channels for this cell. For all cells that are not currently occupied by an agent, these channels are empty.

Table 32: Channels and properties (other agents)

Channel	Property	
34	**Handle**	Identification number of the agent on the cell
35	**Direction**	Orientation in which the agent on the cell is currently directed
36	**Moving**	Indicator whether the agent on the cell is currently moving or not
37	**Speed**	Speed of the agent on the cell
38	**Position fraction**	Position fraction of the agent on the cell
39	**Malfunction**	Number of steps until the malfunction of the agent on the cell ends
40	**Nr malfunctions**	Number of malfunctions that have occurred on the agent on the cell so far
41 - 44	**Distance to target**	Minimum distance from the current cell to the target of the agent on the cell, divided into the four possible orientations (north, east, south, west)

The two channel groups for the observed agent and the other agents are almost identical. One could even come up with the idea to use only one single group instead. However, there is a situation which could then no longer be represented: Agent A1, which has not yet entered the railway network, is to be observed. At the same time, the agent A2 is located on the start cell of A1. Both A1 and A2 should then be expressed on the same cell. Thus, both separate channel groups are required to represent both agents.

The number of channels per cell is constant 44 but the overall state must also have a constant size to enable learning. Therefore, a railway network must always consist of the same number of cells. According to the test cases in Section 6.2, the largest railway network has a height of 150 cells and a width of 150 cells. For this reason, an array with the dimension (44, 150, 150) is to be used as state. For smaller railway networks, the missing space is simply filled with empty cells.

6.8.6 Local Observation

With the previous approach, the railway network as a whole is used as a state, not in the form of an image but in a similar form. The resulting state consists of an array with 990,000 single values for a height and width of 150 cells. A principle of RL states: the larger the state, the more difficult the learning. For this reason, an attempt is made to observe only a local area of the railway network. The basis for such a local observation was provided in the library of the Flatland Environment. Based on this, the final local observation has been developed in the course of this thesis.

6.8.6.1 State Structure

An agent is located on a cell in a certain direction. To build a state for this observed agent, a local observation is made for each possible next cell. As described in 3.2.2.3, there is a maximum of two next cells on each cell and orientation. A state is thus composed of two local observations, the first for the next cell more to the left and the second for the next cell more to the right. If only one next cell exists, the second local observation is empty. Figure 37 contains some examples in which either one or two local observations are made.

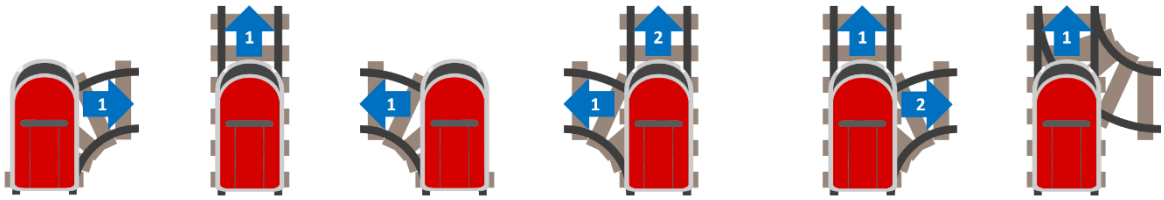


Figure 37: Local observations

6.8.6.2 Observation Structure

A local observation is built in the form of a tree. First, the track is followed to a cell having two next cells. Such a cell is henceforth called a *Fork*. The track section to this first fork corresponds to the root node of the tree. The track section of the root node has various properties which can be used in the observation. In Table 33, the properties that can be observed are listed and explained.

Table 33: Properties of a track section

Property	Description
Track distance	The number of cells of this track section is considered.
Target (minimum total distance)	The minimum distance from this track section to the target position of the observed agent is considered.
Agents in same direction	The number of agents moving in the same direction on this track section is considered.
Agents in opposite direction	The number of agents moving in the opposite direction on this track section is considered.
Another agent (distance)	If there is another agent on this track section, the respective distance is considered.
Possible conflict (distance)	The shortest path of all agents is assumed as their future route. If this assumption would lead to a conflict on this track section, the respective distance is considered.
Target (distance)	If the target position of the observed agent is located on this track section, the respective distance is considered.
Another target (distance)	If the target position of another agent is located on this track section, the respective distance is considered.
Another intersection (distance)	If there is another intersection (except fork) on this track section, the respective distance is considered.
Malfunction	If the observed agent has a malfunction, the remaining number of blocking steps is considered.
Malfunction (others)	The maximum blocking duration of all agents on this track section is considered.
Slowest speed	The slowest speed of all agents in the same direction on this track section is considered.
Agents before start	The number of agents not yet started with a start position on this track section is considered.

After the properties for the root node have been collected, the tree can proceed one level deeper. At the end of each track section there is a fork and thus two child nodes. For these nodes the properties can be calculated again. This can be continued to the desired tree depth. To build up the observation, all nodes of the tree are concatenated at the end.

As an example, Figure 38 contains a part of a railway network with a single agent for which a state is to be determined. From the agent's cell, there is only one next cell and thus the second observation remains empty. The root node of the first observation covers a track section to the first fork. In the second level there are two child nodes up to the next fork. Each of these nodes again has two child nodes due to the fork. This corresponds to an observation with a tree depth of 2, whereby the seven nodes are concatenated to one observation.

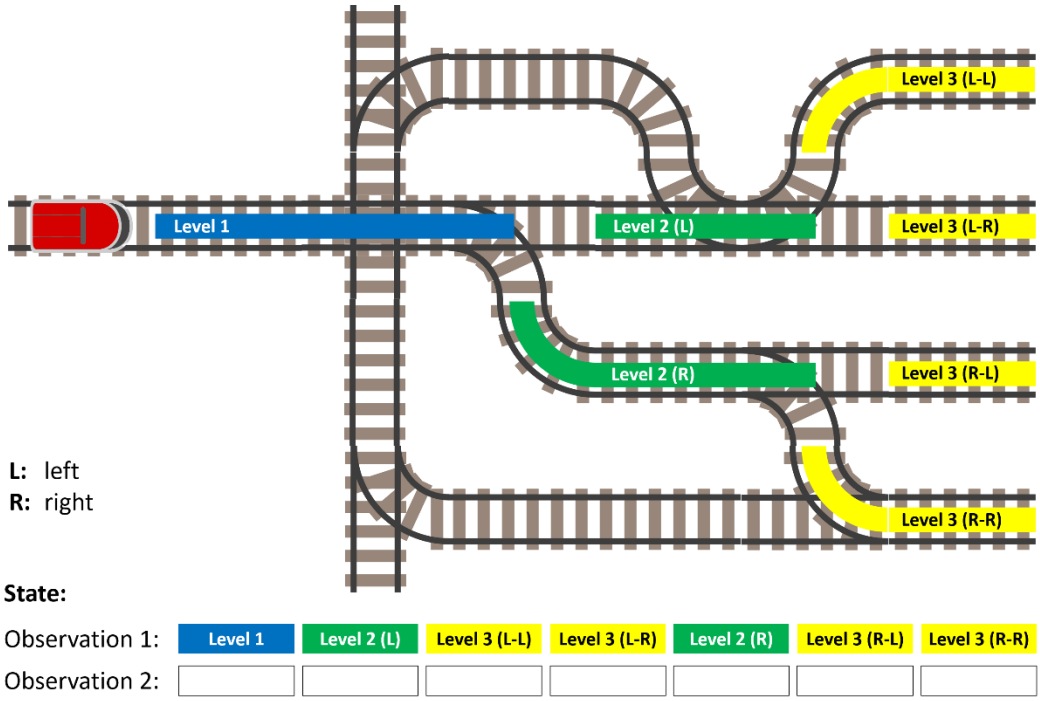


Figure 38: Tree observation (tree depth = 2)

6.8.6.3 Observation Extension

The observation described in the previous section is only one of many possibilities. This observation can be extended with further properties. Another extension concerns the joining paths, which are currently not included in the observation.

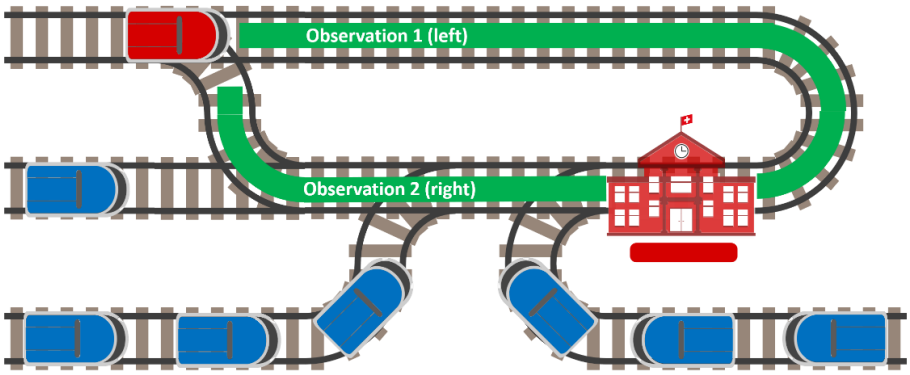


Figure 39: Local observation ignoring joining paths

As an example, Figure 39 illustrates a part of a railway network. Observations are made for the red agent in the upper left corner of which the target position is also marked. The track section covered by observation 2 is definitely shorter, but not necessarily the better choice. There are seven agents which will join the track section of observation 2. Slower agents or malfunctioning agents may cause delays. However, this risk cannot be detected by observation 2, because the joining paths are not considered. To improve the observation, additional properties for the joining paths can be included. For example, the number of joining agents or their lowest speed would be of interest.

6.8.7 Enhancement: Output Mapping

After the RL approach and two different observation types for the state are specified, some improvements can be made. A first enhancement of the RL approach achieves a further reduction of the output size of the neural network.

So far, the output of the neural network has directly corresponded to the different actions of the agent. As mentioned, an agent must always decide between a maximum of two next cells: Either the agent chooses the next cell that is more to the left from its point of view or the next cell that is more to the right. In addition, the agent still has the possibility to stop on its current cell.

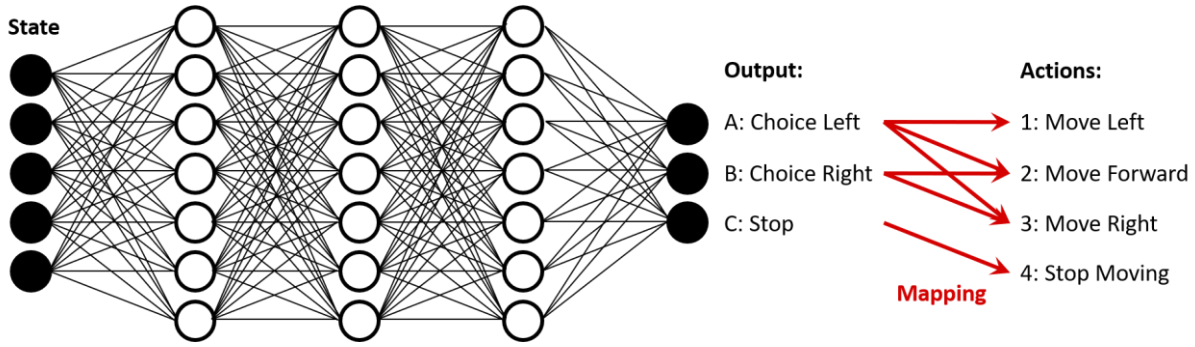
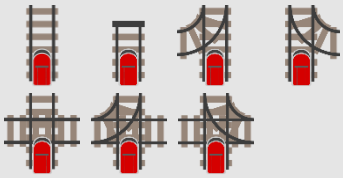


Figure 40: Neural Network with action mapping

With this background knowledge, the output of the neural network can be adapted, as illustrated in Figure 40. The output no longer corresponds directly to the actions. Instead, the output now expresses whether the left track, the right track or a stop should be selected. This choice is then mapped to the actual action depending on the cell type the agent is currently on. In total, there are six different situations to distinguish between in mapping. The corresponding situations and their mappings are listed in Table 34.

Table 34: Action mapping

Current position	A: Choice Left	B: Choice Right	C: Stop
	1: Move Left	- (invalid action)	4: Stop Moving
	2: Move Forward	- (invalid action)	4: Stop Moving
	3: Move Right	- (invalid action)	4: Stop Moving
	1: Move Left	2: Move Forward	4: Stop Moving
	1: Move Left	3: Move Right	4: Stop Moving
	2: Move Forward	3: Move Right	4: Stop Moving

6.8.8 Enhancement: Decisions only

The neural network receives the state as input and calculates the qualities as output. This calculation needs a certain time, which should not be underestimated. As this calculation is executed repeatedly for each choice of the next action, the total calculation time adds up. Another enhancement of the RL approach aims to use the neural network only for real decisions and thus reducing the number of applications of the neural network.

6.8.8.1 Analysis

An analysis of the rail networks has revealed that there are basically only two situations in which a real decision actually has to be taken: If the agent is at a fork or before a join.

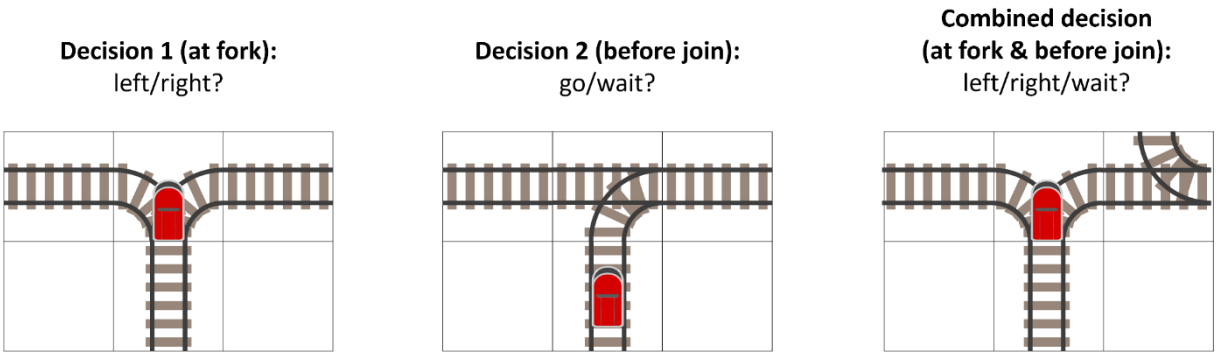


Figure 41: Real decision situations

The situations in which the agent must make a decision are illustrated in Figure 41. The first situation occurs when the agent is directly at a fork. A fork means that there are two possible next cells from the current position and orientation of the agent. In this case, it must be decided whether the left or the right path should be followed. There is no point in waiting: If another agent is approaching in the opposite direction on one route, the other route must be chosen anyway to avoid a deadlock.

The second decision occurs when the agent is on a cell directly before a join. A join refers to a cell where another track path joins the current track of the agent in the same direction. In this case, the agent must decide whether to enter the join on the next cell or wait on the current cell. If another agent is approaching in the same direction from the left side, the agent before the join must decide whether to give right of way (wait) or take it (go). If another agent is approaching in the opposite direction from the right side, the agent before the join must wait to avoid a deadlock.

If the agent is directly at a fork and also before a join, there is a combined decision. First, the agent must decide regarding to the fork (left/right). If the agent has chosen the right side, it must then additionally decide whether to really move to the next cell or to wait.

In all other situations, there is only one next cell for the agent and therefore no Decision 1 (left/right). In addition, moving to this next cell cannot lead to a deadlock in these situations. Thus, it makes no sense to wait and Decision 2 (go/wait) is omitted as well. For the sake of completeness, Figure 42 illustrates some of these situations which do not require a real decision.

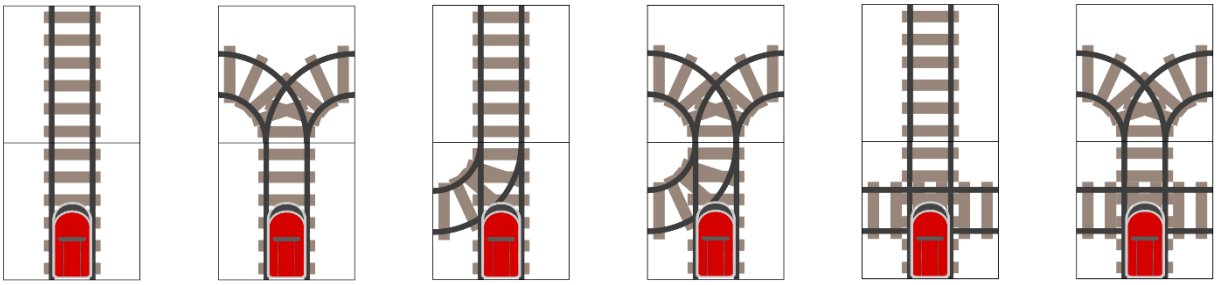


Figure 42: Other situations without real decision

6.8.8.2 Realization

With this background knowledge, the RL approach can be optimized as follows: When the agent has to determine its next action, it is first checked whether the agent is currently in a situation with a real decision. Only if this is the case, the action is selected using the neural network. If not, the agent simply selects the action which leads to the only possible next cell.

This change also requires a change in the reward. With the previously defined rewards, the agent would learn to make as few decisions as possible to reach its target position. However, the path with the fewest real decisions is not necessarily the shortest path. The decisive factor is the number of cells between the real decisions. Therefore, the rewards are adjusted as follows: If a decision is made that causes the agent to reach its target position before the next decision, the maximum number of allowed steps is still returned. For all other decisions, the negated number of cells of the path to the previous decision is returned as the reward.

6.8.8.3 Advantages

This approach has several advantages. The number of applications of the neural network is reduced. This not only saves time in learning, but also leads to more specific learning. With the previous approach, the agent must also learn to drive straight ahead if it can only drive straight ahead. With the optimized approach, the agent no longer must learn such trivial tasks, but can focus on the real decisions.

6.8.9 Enhancement: Deadlock Avoidance

As explained in Section 3.3, deadlocks are a very serious problem in railway networks when multiple agents are on the move. With the current RL approach, no measures are taken to detect deadlocks early and prevent them. Instead, this task is completely left to RL: The agent should learn what action would cause a deadlock and then treat it as an invalid action.

However, it is not a trivial task to detect whether an action will cause a deadlock in the future or not. Under certain circumstances, a bad action can lead to a situation that inevitably results in a deadlock, but only after many time steps. Therefore, a possible simplification of the RL approach is to take over the avoidance of deadlocks in order to ensure that no deadlocks occur using a RL-independent algorithm.

6.8.9.1 Literature Research

In order to check whether such approaches in connection with railway networks are already known in the literature, a literature research was carried out. Inevitably, the two railway deadlock detection methods *Movement Consequence Analysis (MCA)* and *Dynamic Route Reservation (DRR)* are encountered, which were developed by (Pachl, 2007). However, MCA presupposes that the routes of all trains are definitively fixed in advance and cannot be changed. This method is therefore not suitable for the present problem. And also with DRR, the environment is simplified in such a manner that the method cannot easily be extended to the Flatland Environment.

In fact, deadlock detection is an NP-hard problem as proven by (Gawrilow, Klimm, Möhring, & Stenzel, 2012). This means that no complete algorithm has yet been found that can determine with absolute correctness in polynomial time whether a situation leads to a deadlock or not. Nevertheless, different approaches have been investigated and are discussed in the following subsections.

6.8.9.2 Detection by MAPF Algorithms

An algorithm is to be provided which determines whether or not an action would lead to a deadlock. There is an intuitive solution approach to this: One assumes that the action to be checked has already been performed and then checks whether there is a solution for this situation using a complete MAPF algorithm. If there is a solution, the checked action is valid and otherwise invalid.

Unfortunately, all considered complete MAPF algorithms are optimal and therefore not feasible in polynomial runtime. Since the deadlock check is performed for all eligible actions each time the next action is selected, the use of a complete algorithm is not applicable in reasonable time.

Instead of a complete MAPF algorithm, a suboptimal MAPF algorithm could also be used for deadlock detection. However, the considered suboptimal algorithms are incomplete. This means that there are situations in which the algorithm does not find a solution, although there would be one. Thus, it is possible that a situation is recognized as a deadlock even though it is actually not a deadlock at all. But if a situation really leads to a deadlock, this is certainly detected.

6.8.9.3 Detection by Intersection Analysis

As an alternative approach for deadlock detection, the current traffic situation of all railway intersections can be analyzed. The respective algorithm is designed and implemented as follows:

First, it is assumed that the action under review has already been executed. Then, the next upcoming intersection is analyzed. The different traffic situations for a simple or symmetrical switch are shown in Figure 43. In situations 1 - 3, the agent is the only one approaching the intersection from any side. If a second agent would enter from another direction, the situation could still be resolved without deadlock. The situation is different if two agents are approaching the intersection from two different directions. In situation 4, the track on the right is still free. However, it is evident that agent 3 must head right to avoid a deadlock with agent 1. Thus, the arrival of agent 3 is propagated to the next upcoming intersection, as marked with an arrow. Situation 5 is the mirrored version of situation 4 and therefore equivalent in the procedure. In situation 6, one agent comes from each of the two join entrances. It is evident that first both agents must leave downwards before another agent can even enter from below. Although the order of the two agents is not known, the arrival of both agents is propagated to the next intersection.

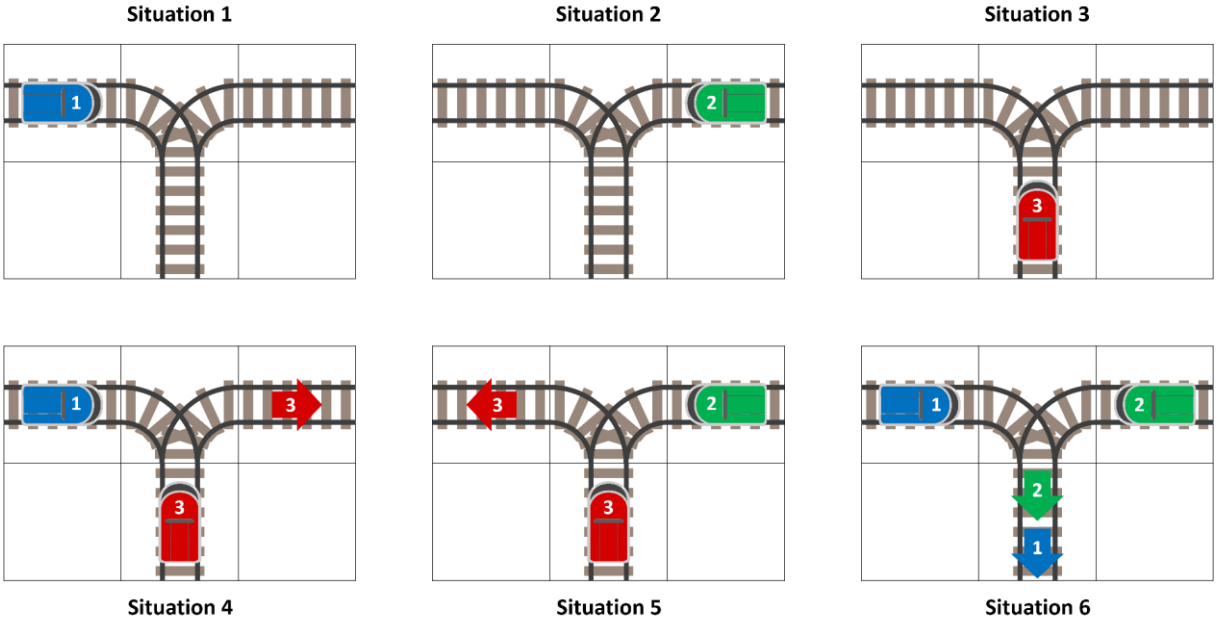


Figure 43: Intersection analysis (simple/symmetrical switch)

If the arrival of one or more agents is propagated to the next intersection, this intersection is analyzed next. If the propagated situation results in a deadlock, the action to be checked is an invalid action. If again an arrival needs to be propagated to a next intersection, the procedure continues in the same way. Only if no more propagation is required, the situation is detected as deadlock-free.

Of course, this procedure is not only possible for simple and symmetrical switches, but also for single and double slip switches. Figure 44 shows all situations on these switches in which propagation to the next intersection is required. In all other situations, there are only two agents and a potential third agent from any direction could still pass the intersection without deadlock. In the situations in the first line containing a single slip switch, it is evident which agents are to be propagated in the remaining direction.

In the situations in the second line containing a double slip switch, it is evident that at least one agent must head in the remaining direction. However, it is not yet known which agent is concerned. Therefore, a placeholder is propagated.

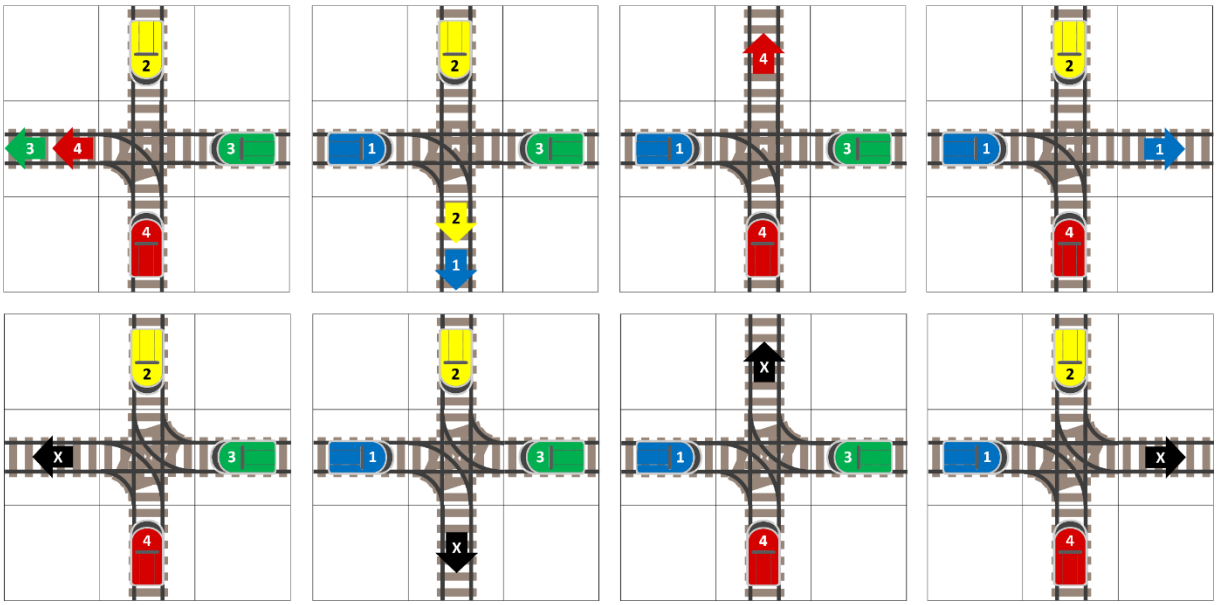


Figure 44: Intersection analysis (single/double slip switch)

With the help of the propagating intersection analysis described above, a major part of the situations leading to a deadlock can be detected. If a situation is detected as deadlock, it is certainly a deadlock. However, there are some deadlock situations, especially caused by tailbacks, which cannot be detected. Thus, it is possible that an action is recognized as valid although it causes a deadlock.

6.8.9.4 Detection by Complete Path Reservation

Another approach for deadlock detection relies not on existing MAPF algorithms but on an existing VRSP algorithm, namely on *Complete Path Reservation (CPR)* described in Section 6.7. The procedure is quite simple: To check whether an action leads to a deadlock, it is tried to find a non-reserved path from the corresponding next cell to the target position. If this is successful, the action certainly does not lead to a deadlock. If not, the action is assumed to lead to a deadlock.

This approach is designed to be robust and safe. Thus, it is possible that a situation is recognized as a deadlock although it is actually not a deadlock at all. But if a situation really leads to a deadlock, this is certainly detected.

6.8.9.5 Conclusion

In the previous sections, different approaches for deadlock detection are described, which completely take over deadlock detection or simplify the learning of deadlock situations for the RL approach. Deadlock detection by a complete MAPF algorithm would detect the presence or absence of a deadlock completely correctly. However, it is not reasonable to use this approach due to the non-polynomial runtime.

If an incomplete MAPF algorithm is used instead, all deadlocks will be detected and some additional deadlock-free situations will be falsely recognized as deadlocks. The same applies to deadlock detection by CPR. Furthermore, deadlock detection by intersection analysis can also be used. However, some deadlock situations are not detected as deadlock with this approach. These situations still have to be learned by RL.

6.8.10 Enhancement: State Partitioning

A further enhancement option is related to the state as a local observation. As described in Section 6.8.6, there are two local observations, one for the next cell more to the left and one for the next cell more to the right. These two observations are then sequentially strung together as the state of the environment. In Figure 45, two railway networks are shown, whereby the two local observations are marked for the red agent. At first glance, the two rail networks may appear to be different, but in principle they are simply mirrored. Observation 1 on the left-hand railway network corresponds to observation 2 on the right-hand railway network. This can also be seen in the state where both observations are interchanged.

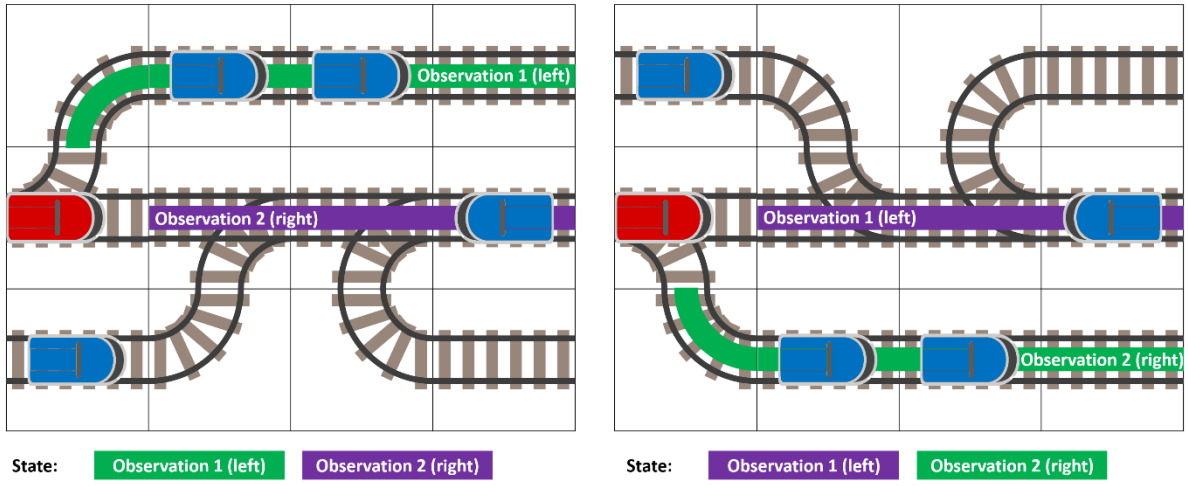


Figure 45: Local observations on mirrored railway networks

If an agent is in the left situation, it learns whether it should go left or straight ahead. But this depends exactly on the corresponding state. The agent does not learn that it should simply take the mirrored action for a mirrored state. This circumstance can be improved for the RL approach.

In the current RL approach, the two observations are concatenated into a one-dimensional array, as shown in Figure 46. This state is then passed as input to the neural network. However, the neural network receives an array of values and does not know that the first half and the second half of the state actually have the same structure and represent two different observations. Therefore, the neural network must calculate the qualities of the actions from the whole concatenated array.

As an improvement, the two observations are instead combined into a two-dimensional array. Now, the neural network can first be applied to observation 1, from which the quality for the decision to the left is calculated. Then, the neural network is applied to observation 2 to also calculate the quality for decision to the right. This reduces both the size of the input and the size of the output of the neural network. At the same time, the neural network only calculates the quality of the action directly related to the corresponding observation and not the qualities of several actions based on both observations.

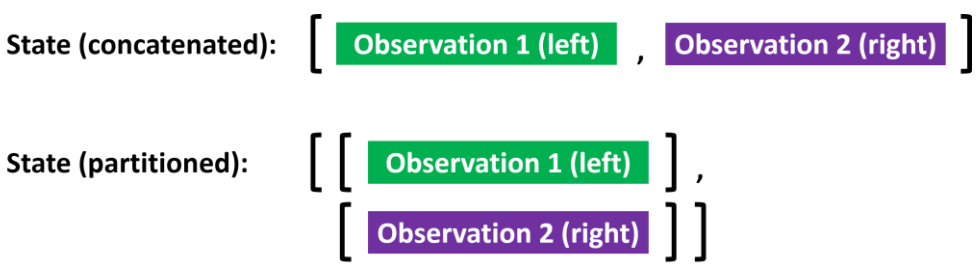


Figure 46: State partitioning

6.9 Probabilistic Programming

In addition to the approaches considered so far, the so-called *Probabilistic Programming* is a potential approach to solve the VRSP problem as well. As the name implies, this approach is all based on probabilities. In this context, a statistical model is to be specified for a concrete problem.

Applied to the Flatland Environment, the environment is expressed in terms of probabilities, for example for the next action to be taken or the occurrence of malfunctions. As with RL, this probability model provides the knowledge of an agent that is subsequently applied to the current situation to select the next action. The difference between probabilistic programming and RL concerns the way this knowledge is built up: While knowledge is learned by trial and error in RL, it is specified by probabilities in this approach.

Since the two approaches have great similarities, it was decided after a brief assessment not to pursue this approach any further. Instead, the focus will be put more on reinforcement learning.

6.10 Conclusion

At the end of Round 2 of the Flatland Challenge, the following conclusion can be drawn: The fundamental problem of this final round is known as Vehicle Rescheduling Problem (VRSP). There are existing approaches to solve this problem. However, there are also novel approaches, especially reinforcement learning, which can be applied to the problem. It is necessary to check to what extent the novel approaches can compete with existing approaches or whether they even produce better results. The corresponding results are discussed in Section 7.

The collection of approaches described in this section does not claim to be complete. There are other approaches for VRSP, which are not discussed in this report.

7 Results

Many different approaches to the two fundamental problems MAPF and VRSP are described and discussed in the previous sections. In the course of this thesis, all approaches were implemented and evaluated using the corresponding test cases of the Flatland Challenge. This section describes how the evaluation of the different approaches was carried out and reviews their results.

7.1 Hardware Specifications

All approaches were tested on the same workstation. The corresponding hardware specifications are provided in Table 35. It is a reasonably powerful computer and sufficient for evaluation in this context. However, there are much more high-performance computers that would give a noticeable performance increase. In particular, more powerful graphics options are recommended for the computing-intensive learning process in RL.

Table 35: Technical specifications

Workstation	DELL Precision 3630 Tower CTO BASE
Processor Options	Intel i7-8700K, 6 Core, 12MB Cache, 3.7GHz, 4.7Ghz Turbo
Memory Options	32GB DDR4-UDIMM (2 x 16GB), 2.666MHz, Non-ECC
Storage Options	M.2 PCIe NVMe SSD, 256GB 3.5'' SATA, 1TB, 7200 RPM
Graphics Options	NVIDIA GeForce GTX 1060, 6GB, 3 DP, HDMI, DVI-D

7.2 Railway Network Encoding

All considered encodings for a railway network were implemented. Finally, the representation in form of a cell orientation graph (see Section 4.1.2) was used consistently for the evaluation of the approaches. Thus, all approaches had the same basic conditions.

7.3 Multi-Agent Path Finding (MAPF)

The first fundamental problem concerns the initial search of conflict-free paths for all involved agents. In the following subsections, the different MAPF algorithms are reviewed and evaluated.

7.3.1 Linear Programming (LP)

In linear programming, a test case is formulated as a mathematical optimization model. This optimization problem is then processed using existing LP solvers to find an optimal solution. For the model formulation, the two different Python libraries *mip* and *PuLP* were used. Both libraries use a solver called *Coin-or branch and cut (CBC)* to solve the model.

The test cases given in Round 1 of the Flatland Challenge (see Section 5.2) were solved using both libraries. In Table 36, the runtimes (in seconds) of the two libraries for the different test cases are compared. It becomes evident that simple test cases can be solved with LP. But when the railway network becomes more complex and the number of agents increases, this approach quickly reaches its limits. After 10 minutes of calculation time, the process was aborted which results in a timeout. This has already happened in test case 2.0 with five agents and over 60 track cells.

Table 36: Results for linear programming

Test	mip (CBC)	PuLP (CBC)	Test	mip (CBC)	PuLP (CBC)
0.0	0.35225	0.31807	**5.0**	*timeout*	*timeout*
0.1	0.28387	2.25091	**5.1**	*timeout*	*timeout*
1.0	9.58017	1.33192	**6.0**	*timeout*	*timeout*
1.1	115.14047	198.57655	**6.1**	*timeout*	*timeout*
2.0	*timeout*	*timeout*	**7.0**	*timeout*	*timeout*
2.1	*timeout*	*timeout*	**7.1**	*timeout*	*timeout*
3.0	*timeout*	*timeout*	**8.0**	*timeout*	*timeout*
3.1	*timeout*	*timeout*	**8.1**	*timeout*	*timeout*
4.0	*timeout*	*timeout*	**9.0**	*timeout*	*timeout*
4.1	*timeout*	*timeout*	**9.1**	*timeout*	*timeout*

7.3.2 Constraint-based Search (CBS)

With constraint-based search, an initial solution is first created containing the shortest path for each agent. In this solution, there are often conflicts between two or more agents. The first conflict is then considered, and two successor nodes are created: In the first node, the shortest path avoiding this conflict is searched for one of the two agents. In the second node, the other agent is handled. Then, the process is continued with the node that promises the best solution. The other node is put aside for later consideration. If there are conflicts in the next node, the procedure is repeated until a conflict-free solution is found. This is the optimal solution.

Often, it happens that multiple nodes promise the same objective value (makespan). In this case, it is necessary to decide which additional criteria are used for prioritization. For test purposes, the following four prioritizations were examined:

A) If multiple nodes have the same makespan, the node with the **highest** number of conflicts is taken. If there are still multiple nodes of choice, the node that was created **first** is used.

B) If multiple nodes have the same makespan, the node with the **highest** number of conflicts is taken. If there are still multiple nodes of choice, the node that was created **last** is used.

C) If multiple nodes have the same makespan, the node with the **lowest** number of conflicts is taken. If there are still multiple nodes of choice, the node that was created **first** is used.

D) If multiple nodes have the same makespan, the node with the **lowest** number of conflicts is taken. If there are still multiple nodes of choice, the node that was created **last** is used.

The test cases given in Round 1 of the Flatland Challenge (see Section 5.2) were solved using CBS and the prioritizations described above. In Table 37, the resulting runtimes (in seconds) and the number of processed nodes are compared. After 10 minutes of calculation time, the process was aborted which results in a timeout.

It is shown that the selection of the prioritization has a decisive factor on the duration of the algorithm. As an example, consider test case 6.1: With prioritization B, an optimal solution is found after six nodes already. With the prioritizations C and D, it takes considerably longer (70 and 73 nodes). But with prioritization A, a timeout occurs after 10 minutes and 46,424 nodes processed so far. The same pattern can be seen in other test cases as well.

Table 37: Results for constraint-based search

Test	CBS (A)		CBS (B)		CBS (C)		CBS (D)	
	Runtime	Nodes	Runtime	Nodes	Runtime	Nodes	Runtime	Nodes
0.0	0.00006	1	0.00006	1	0.00005	1	0.00005	1
0.1	0.00004	1	0.00004	1	0.00004	1	0.00004	1
1.0	0.00029	2	0.00024	2	0.00025	2	0.00027	2
1.1	0.00012	1	0.00013	1	0.00011	1	0.00013	1
2.0	0.00122	7	0.00110	7	0.00109	7	0.00115	7
2.1	0.00025	1	0.00023	1	0.00019	1	0.00018	1
3.0	0.00458	5	0.02175	27	0.01204	19	0.01533	23
3.1	0.00283	4	0.00139	2	0.00198	3	0.00139	2
4.0	62.59917	14495	0.01252	13	0.27390	272	0.25813	257
4.1	*timeout*	*77133*	*timeout*	*78521*	15.46160	9640	18.63132	11003
5.0	0.01300	7	0.01129	8	0.05399	42	0.06313	54
5.1	0.01650	7	0.02905	10	0.14201	83	0.17150	99
6.0	5.79081	449	0.05049	10	0.07581	22	0.10247	28
6.1	*timeout*	*46424*	0.01685	6	0.17409	70	0.18087	73
7.0	1.10606	86	0.85441	81	*timeout*	*49494*	*timeout*	*48995*
7.1	1.06890	78	0.88645	63	*timeout*	*43430*	*timeout*	*41939*
8.0	0.00315	1	0.00293	1	0.00302	1	0.00303	1
8.1	0.00496	1	0.00478	1	0.00485	1	0.00486	1
9.0	5.01924	88	30.76673	156	*timeout*	*17011*	*timeout*	*16439*
9.1	3.42885	56	6.75903	86	*timeout*	*14595*	*timeout*	*14289*

7.3.3 Operator Decomposition (OD) & Independence Detection (ID)

With the initial A* approach and operator decomposition (OD), an optimal solution is built up step by step. In the A* approach, one step includes the next actions of all agents. In OD, one step only includes the next action of the next agent in the sequence. However, the MAPF problem can be solved with both A* and OD. As a further performance optimization, agents can be grouped together using independence detection (ID).

The test cases given in Round 1 of the Flatland Challenge (see Section 5.2) were solved using the approaches A*, OD, and OD+ID. In Table 38, the runtimes (in seconds) of these approaches for the different test cases are compared. After 10 minutes of calculation time, the process was aborted which results in a timeout.

As already expected, performance differences become apparent. With the A* approach, simple test cases can be solved. But when the railway network becomes more complex and the number of agents increases, this approach quickly reaches its limits. With OD, the performance can be increased, resulting in fewer timeouts. If ID is additionally used, the performance is increased even more. Nevertheless, there are still some test cases which cannot be solved within the time limit.

Table 38: Results for A, OD, and OD+ID*

Test	A*	OD	OD+ID
0.0	0.00013	0.00016	0.00005
0.1	0.00016	0.00021	0.00006
1.0	0.00081	0.00054	0.00056
1.1	0.00083	0.00058	0.00013
2.0	0.11453	0.00200	0.00104
2.1	0.00568	0.00105	0.00020
3.0	0.48230	0.31263	0.01082
3.1	*timeout*	0.23739	0.00358
4.0	*timeout*	*timeout*	*timeout*
4.1	*timeout*	*timeout*	364.95296
5.0	*timeout*	232.09726	0.12862
5.1	*timeout*	*timeout*	0.06104
6.0	*timeout*	*timeout*	0.05177
6.1	*timeout*	*timeout*	0.07653
7.0	*timeout*	*timeout*	*timeout*
7.1	*timeout*	*timeout*	0.39294
8.0	2.11119	0.07348	0.00358
8.1	2.29850	0.13422	0.00564
9.0	*timeout*	*timeout*	*timeout*
9.1	*timeout*	*timeout*	*timeout*

7.3.4 Suboptimal Algorithms

As the results of the approaches evaluated so far have shown, there are considerable differences between the individual algorithms but also between the different test cases. There are test cases where an optimal solution is found very fast. But there are also many test cases which lead to timeouts in different algorithms. However, train traffic is a real-time system that depends on very short calculation times. Therefore, suboptimal approaches are reviewed and evaluated in the following subsections. In addition to performance, it is of central importance of how close to the optimal solution the suboptimal results are.

7.3.4.1 Optimal Anytime Algorithm (OAA)

The optimal anytime algorithm uses the OD+ID approach in a slightly modified form. The algorithm first calculates a correct but bad solution. This solution is then used as a lower bound to continuously discover better solutions using OD+ID until an optimal solution is found or the timeout is reached.

The test cases given in Round 1 of the Flatland Challenge (see Section 5.2) were solved using OAA. In Table 39, the runtimes (in seconds) and the difference to the optimal objective value (makespan) are compared for all test cases. After 10 minutes of calculation time, the process was aborted and the current best solution was taken. It becomes apparent that an optimal solution was found in most cases. A total of six test cases resulted in a timeout. In four of these aborted test cases, an optimal solution has already been found within the time limit, but not yet recognized as optimal. Nevertheless, this suboptimal approach still causes timeouts and is therefore not suitable for a real-time system.

Table 39: Results for optimal anytime algorithm

Test	OAA runtime	OAA makespan		Optimal makespan
0.0	0.00005	8	*(=)*	8
0.1	0.00006	11	*(=)*	11
1.0	0.00128	10	*(=)*	10
1.1	0.00012	11	*(=)*	11
2.0	0.00168	11	*(=)*	11
2.1	0.00024	11	*(=)*	11
3.0	0.01600	28	*(=)*	28
3.1	0.00603	26	*(=)*	26
4.0	*timeout*	30	*(=)*	30
4.1	*timeout*	33	*(+4)*	29
5.0	0.11072	49	*(=)*	49
5.1	0.18168	53	*(=)*	53
6.0	0.10445	64	*(=)*	64
6.1	0.16499	61	*(=)*	61
7.0	*timeout*	75	*(=)*	75
7.1	*timeout*	80	*(+1)*	79
8.0	0.00351	82	*(=)*	82
8.1	0.00559	130	*(=)*	130
9.0	*timeout*	147	*(=)*	147
9.1	*timeout*	165	*(=)*	165

7.3.4.2 Prioritized Planning (PP)

With prioritized planning, one agent after the other is greedily assigned the shortest path that does not conflict with the existing paths. The prioritization of the agents, i.e. the order in which they are handled, can be defined according to any criteria. For test purposes, the following four prioritizations were examined:

A) The agents are prioritized from the **lowest to the highest identification number**. This means that no prior analysis of the agents is done at all.

B) The agents are prioritized from the **highest to the lowest distance** of the shortest path. In case of equality, the affected agents are prioritized according to the **lower identification number**.

C) The agents are prioritized from the **lowest to the highest distance** of the shortest path. In case of equality, the affected agents are prioritized according to the **higher identification number**.

The test cases given in Round 1 of the Flatland Challenge (see Section 5.2) were solved using PP and the prioritizations described above. In Table 40, the resulting runtimes (in seconds) and the difference to the optimal objective value (makespan) are compared for all test cases. First of all, it can be seen that this approach has the best results so far with regard to the runtime. All test cases are solved within less than two seconds. Another aspect is the quality of the solution. It is shown that the prioritization has an influence on the solution. The solutions of the best considered prioritization (B) requires in total only 4 time steps more than the optimal solution

for all test cases (+0.37%). With the worst considered prioritization (C), it takes 21 time steps more in total (+1.94%). These small differences are very respectable, especially when the quality of the solutions is compared to the required runtime.

Table 40: Results for prioritized planning

Test	Optimal makespan	PP (A)		PP (B)		PP (C)	
		Runtime	Diff	Runtime	Diff	Runtime	Diff
0.0	8	0.00006	(=)	0.00008	(=)	0.00008	(=)
0.1	11	0.00007	(=)	0.00010	(=)	0.00010	(=)
1.0	10	0.00020	+1	0.00027	(=)	0.00026	+1
1.1	11	0.00019	(=)	0.00024	(=)	0.00025	(=)
2.0	11	0.00038	(=)	0.00053	(=)	0.00065	+2
2.1	11	0.00032	(=)	0.00042	(=)	0.00043	(=)
3.0	28	0.00333	(=)	0.00443	(=)	0.00434	(=)
3.1	26	0.00204	(=)	0.00216	(=)	0.00226	(=)
4.0	30	0.00596	(=)	0.00748	(=)	0.00556	+8
4.1	29	0.00512	+4	0.00518	+4	0.00649	+4
5.0	49	0.00835	(=)	0.01192	(=)	0.00961	(=)
5.1	53	0.02027	+2	0.01502	(=)	0.02167	+2
6.0	64	0.02123	(=)	0.02240	(=)	0.01548	(=)
6.1	61	0.01807	+3	0.01988	(=)	0.01867	+3
7.0	75	0.19581	(=)	0.25005	(=)	0.20453	(=)
7.1	79	0.27885	+1	0.36779	(=)	0.27110	+1
8.0	82	0.01642	(=)	0.01746	(=)	0.01788	(=)
8.1	130	0.02390	(=)	0.02567	(=)	0.02663	(=)
9.0	147	1.38655	(=)	1.63163	(=)	1.01304	(=)
9.1	165	1.53378	(=)	1.68559	(=)	1.48069	(=)
Total Diff	**1080**		**+11**		**+4**		**+21**

7.4 Vehicle Rescheduling Problem (VRSP)

The second and final problem concerns the replanning of paths for all involved agents when malfunctions occur. In researching various approaches, especially methods with decentralized control were identified as potentially suitable approaches. In the following subsections, the different VRSP approaches are reviewed and evaluated.

7.4.1 Complete Path Reservation (CPR)

With complete path reservation, a robust approach is applied to avoid deadlocks. If an agent wants to perform an action, it must first reserve the complete path from its current position to its target position in the appropriate direction. Only if this is successful, the action is allowed to be executed. In this way, it can be prevented that two agents can simultaneously enter a cell from a different direction. Again, a prioritization is required, in which order the agents can select their next action and thereby reserve a path. For test purposes, the following four prioritizations were examined:

A) First, agents located at a position where more other agents want to start are prioritized. Then, agents with the higher speed are prioritized. Finally, agents with shorter remaining paths are prioritized.

B) First, agents with shorter remaining paths are prioritized. Then, agents with the higher speed are prioritized. Finally, agents located at a position where more other agents want to start are prioritized.

C) First, agents with the higher speed are prioritized. Then, agents with shorter remaining paths are prioritized. Finally, agents located at a position where more other agents want to start are prioritized.

D) Agents with a lower identification number are prioritized.

The test cases given in Round 2 of the Flatland Challenge (see Section 6.2) were solved using CPR and the prioritizations described above. The results are listed and compared in Table 41. A maximum number of allowed time steps was defined for each test case. If all agents (100%) reached their target positions within this limit, the number of steps used for this is stated. If this was not the case, the step limit is given in brackets, and the percentage of agents that reached their target positions within this limit is stated.

Table 41: Results for complete path reservation

Test	CPR (A)		CPR (B)		CPR (C)		CPR (D)	
	Steps	Agents	Steps	Agents	Steps	Agents	Steps	Agents
0.0	379	100.00%	379	100.00%	379	100.00%	384	100.00%
0.1	527	100.00%	527	100.00%	527	100.00%	592	100.00%
1.0	594	100.00%	(600)	98.75%	(600)	95.00%	(600)	96.25%
1.1	(600)	93.75%	(600)	87.50%	(600)	87.50%	(600)	90.00%
2.0	627	100.00%	(720)	98.75%	627	100.00%	691	100.00%
2.1	627	100.00%	593	100.00%	627	100.00%	521	100.00%
3.0	(960)	88.75%	(960)	62.50%	(960)	60.00%	(960)	68.75%
3.1	(960)	88.75%	(960)	98.75%	(960)	93.75%	(960)	92.50%
4.0	752	100.00%	713	100.00%	863	100.00%	732	100.00%
4.1	(960)	85.00%	(960)	80.00%	(960)	96.25%	(960)	86.25%
5.0	(1120)	91.25%	(1120)	92.50%	(1120)	92.50%	(1120)	77.50%
5.1	(1120)	92.50%	(1120)	90.00%	(1120)	90.00%	(1120)	90.00%
6.0	(1760)	75.00%	(1760)	84.00%	(1760)	60.00%	(1760)	67.00%
6.1	(1760)	68.00%	(1760)	65.00%	(1760)	66.00%	(1760)	52.00%
7.0	(1600)	84.00%	(1600)	87.00%	(1600)	81.00%	(1600)	73.00%
7.1	(1600)	88.00%	(1600)	70.00%	(1600)	73.00%	(1600)	68.00%
8.0	(1760)	65.00%	(1760)	59.00%	(1760)	58.00%	(1760)	46.50%
8.1	(1760)	52.00%	(1760)	68.50%	(1760)	59.00%	(1760)	69.00%
9.0	(2560)	64.50%	(2560)	56.50%	(2560)	58.50%	(2560)	66.50%
9.1	(2560)	78.50%	(2560)	68.00%	(2560)	71.50%	(2560)	71.00%
Ø		**85.75%**		**83.34%**		**82.10%**		**80.71%**

As expected, the various prioritizations lead to different results. The biggest discrepancy is in test case 3.0, where the difference between the results of prioritizations A and B is more than 28 percentage points. In total, it can be observed that this approach works well for simpler test cases. However, if the complexity of the railway networks and the number of agents increases, the results become noticeably worse. Across all test cases, an average of 85.75% of all agents reach their target position within the step limit using the best prioritization (A).

7.4.2 Reinforcement Learning (RL)

With reinforcement learning, a novel approach to solve the VRSP problem has been investigated. Thereby, the agent should learn independently by long-term trial and error which action leads to the greatest success in which situation. In this novel field, there are countless methods to tackle the matter. The approach used in this thesis is only one of these methods.

7.4.2.1 Initial Learning

In order to initially test the functioning of RL in connection with the Flatland Environment, the parameters were kept to a minimum. The corresponding parameters are described in Table 42.

Table 42: Initial RL parameters

Parameters	Description
Agents	There is only one single agent in the entire railway network.
Grid	A small grid with a height of 35 cells and a width of 35 cells is used.
Speed	The single agent has the highest speed 1.
Episodes	There are 15,000 learning episodes. An episode corresponds to the simulation of a test case.
Epsilon	The learning starts with an epsilon of 1, which is multiplied by a factor of 0.998 after each episode to reduce it down to a minimum value of 0.005.
Discount factor	A discount factor of 0.99 is used to consider the following rewards as well. This is important to recognize the reaching of the target position in a later action.
Test cases	Based on the other parameters, a new environment consisting of a railway network and an agent is randomly generated for each episode.

From these parameters, it can be seen that the initial question is whether an agent in a railway network can learn to move to its target position by the shortest route. Since there is only one agent on the railway network, no deadlocks can occur. This should be an easy task compared to the final VRSP problem. Thus, this initial learning is only about whether RL basically works in the Flatland Environment.

7.4.2.2 Observations

In a first step, it was examined whether both state approaches work: global observation and local observation. With the help of local observation, learning works flawlessly. In the overwhelming majority of tested environments, the agent finds and the shortest path and moves directly to its target position.

In contrast, the attempt with the global observation leads to a failure. As already mentioned, a disadvantage of global observation is that it results in a very large state. After only a few episodes, not enough memory can be allocated and the process is aborted. By reducing the number of observation channels and downsizing the grid, the abort can be delayed, but still not prevented. For this reason, the RL approach with global observation could no longer be pursued. To continue, a massively more powerful computer would be needed.

7.4.2.3 Enhancements

The further attempts concerning RL could only be continued with local observation. However, there are several enhancements to optimize learning:

- Output Mapping
- Decisions only
- State Partitioning

These enhancements were applied to initial learning, both individually and in combination. The results were positive so that the enhancements could be further pursued.

7.4.2.4 Final Learning

With the help of the initial learning, it could be successfully proven that RL can basically be applied to the Flatland Environment. Nevertheless, the initial learning is not yet useful for the final VRSP task, namely collaborative path finding despite different speeds and malfunctions. Therefore, the parameters were adjusted accordingly as described in Table 43. Unmodified parameters are not listed again. For test purposes, two alternative configurations were used, which differ in the number of agents and in the dimension of the railway network.

Table 43: Extended RL parameters

Parameters	Configuration A	Configuration B
Agents	50 agents	80 agents
Grid	Height: 35 cells Width: 20 cells	Height: 35 cells Width: 35 cells
Speed	All agents are equally distributed among the four different speed profiles.	
Malfunctions	The properties of the official test cases are used: The malfunction rate is 12,000 and a malfunction takes at least 20 and at most 50 time steps.	

In comparison to the initial parameters, there is not just one agent operating on the railway network, but many more. Due to this modification, deadlocks between several agents can occur, which must be prevented.

7.4.2.5 Final Observation

Due to the findings during initial learning, the approach with a global observation had to be discarded. However, the local observation approach did very well in initial learning. Thus, local observation was used for this final learning, optimized with all discussed enhancements.

7.4.2.6 Deadlock Avoidance

In the course of this thesis, different approaches to detect deadlocks were considered. However, complete deadlock detection cannot be achieved in polynomial time. There are other algorithms that have a polynomial runtime but are incomplete. Some of them are safety conscious: All deadlock situations are correctly detected as deadlock. In addition, some deadlock-free situations are falsely detected as deadlocks. Other algorithms behave in the opposite way: If a situation is detected as a deadlock, it is really a deadlock. But it is also possible that a deadlock is not recognized as a deadlock at all.

However, an important factor for the use of deadlock detection in RL is the runtime. Learning takes a long time and each additional calculation costs valuable time. Based on this consideration, it was decided to use deadlock detection by complete path reservation. On the one hand, this approach has the best performance of all considered approaches. On the other hand, all deadlock situations are certainly detected as deadlock. This eliminates the need for the agent to learn deadlock situations independently in a time-consuming manner.

7.4.2.7 Results

A final learning process was performed with both configurations described. The configurations correspond to the properties of rather smaller test cases. Nevertheless, the learning with the specified parameters and the existing hardware lasted over two weeks per configuration. After learning from randomly generated test cases, the acquired knowledge was applied to the test cases given in Round 2 of the Flatland Challenge (see Section 6.2). The results are listed and compared in Table 44, with the same structure as for Table 41.

Table 44: Results for reinforcement learning

Test	RL (A)		RL (B)	
	Steps	Agents	Steps	Agents
0.0	374	100.00%	412	100.00%
0.1	(600)	88.00%	(600)	90.00%
1.0	(600)	98.75%	(600)	98.75%
1.1	(600)	82.50%	(600)	85.00%
2.0	(720)	33.75%	(720)	90.00%
2.1	(720)	60.00%	(720)	88.75%
3.0	(960)	56.25%	(960)	56.25%
3.1	(960)	71.25%	(960)	73.75%
4.0	945	100.00%	(960)	98.75%
4.1	(960)	81.25%	(960)	75.00%
5.0	(1120)	77.50%	(1120)	76.25%
5.1	(1120)	58.75%	(1120)	73.75%
6.0	(1760)	77.00%	(1760)	75.00%
6.1	(1760)	57.00%	(1760)	57.00%
7.0	(1600)	51.00%	(1600)	55.00%
7.1	(1600)	57.00%	(1600)	67.00%
8.0	(1760)	40.00%	(1760)	51.00%
8.1	(1760)	51.00%	(1760)	53.50%
9.0	(2560)	34.00%	(2560)	46.00%
9.1	(2560)	33.50%	(2560)	32.50%
Ø		**65.43%**		**72.16%**

As can be observed, the differences between the configurations have a great influence on the success of learning. With configuration B, an average of 72.16% of all agents reach their target position within the step limit across all test cases. This is nearly 7 percentage points more than with configuration A. However, it is also over 13 percentage points less than with the alternative CPR approach. The comparison between the configurations A and B suggests that the results can be further improved by increasing the configuration parameters. However, this would also massively increase the learning time.

8 Conclusion

In this report, different approaches for both the underlying Multi-Agent Path Finding (MAPF) and the related Vehicle Rescheduling Problem (VRSP) were investigated with respect to railway traffic. The research was conducted based on the Flatland Challenge hosted by Swiss Federal Railways (SBB), which provided a suitable framework to address these problems.

8.1 Multi-Agent Path Finding (MAPF)

The aim of multi-agent path finding is to find conflict-free paths for any number of trains in a railway network to their respective destinations. There are various approaches to solve this task. With some discussed approaches, such as the Linear Programming (LP), Constraint-based Search (CBS) and Operator Decomposition & Independence Detection (OD+ID), it is even possible to find an optimal solution for this problem. However, this optimality also entails the drawback of a non-polynomial runtime. An evaluation based on test cases of the Flatland Challenge revealed considerable performance differences between the various approaches. The LP approach respectively the LP solver used for this purpose achieved very limited results. Only 20% of all test cases could be solved using LP within a time limit of ten minutes. The OD+ID approach seems to be much better suited for MAPF in the given context, as around 80% of all test cases could be solved within short time. The CBS approach achieved the best performance among the optimal approaches with a timeout rate of only 5% for the best considered prioritizations. However, the prioritization of the nodes is crucial in this approach.

Since decisions in railway traffic have to be made in real time, a non-polynomial runtime is not well suited for this purpose. For this reason, suboptimal approaches to MAPF, such as the Optimal Anytime Algorithm (OAA) and Prioritized Planning (PP) were additionally explored. The OAA approach is a modified form of OD+ID, which continuously searches for an even better solution until the process is aborted. Thus, the calculation can still take a relatively long time. A better alternative is the PP approach, which very quickly returns a solution for all test cases that is within a tolerable distance from the optimal solution. Therefore, it is appropriate to use a suboptimal algorithm such as PP for an efficient solving of the MAPF problem.

8.2 Vehicle Rescheduling Problem (VRSP)

The vehicle rescheduling problem is about route replanning when a train malfunctions in order to keep the delay of all trains as low as possible. The intuitive approach for this purpose is to perform a new planning using a MAPF algorithm in case of a malfunction. However, this is not practical in the Flatland Environment, as the MAPF algorithm often causes a deadlock between trains when a malfunction occurs. Instead, the Complete Path Reservation (CPR) approach that is robust against deadlocks was introduced in this thesis. With the help of path reservations, it is ensured that all cells are only travelled in one direction at the same time and thus prevent deadlocks. The robustness entails that some additional time steps have to be taken for reliability reasons. Nevertheless, the evaluation based on the test cases of the Flatland Challenge showed that an average of 85.75% of all trains reach their destination within a limited number of time steps.

8.2.1 Reinforcement Learning

A novel approach for the VRSP is Reinforcement Learning (RL) from the field of machine learning. Thereby, an agent learns independently by trial and error which action is most promising in which situation. In the course of the thesis, different RL approaches and additional enhancements were investigated. Finally, the most promising knowledge was built up with a local observation of the environment extended by output mapping, decision differentiation, state partitioning and deadlock prevention. The best attempted learning configuration resulted in an average success rate of 72.16% for the test cases of the Flatland Challenge. This value is significantly below the success rate of the CPR approach. However, the potential of RL as a novel approach is evident. The achieved rate is very promising considering that the exploration of RL was limited by the given hardware and the short project duration.

8.2.2 Deadlock Detection

An important factor in VRSP is the avoidance of deadlocks. In this report, different approaches to deadlock detection were investigated. A complete detection of deadlocks cannot be performed in polynomial runtime. In alternative approaches, either some deadlock-free situations are falsely detected as deadlocks or some of the actual deadlock situations are falsely not detected as deadlocks. Thus, it seems to be a reasonable approach to learn the detection of deadlocks completely or partially by means of RL. Nevertheless, in critical systems such as railway traffic, it is not yet safe to fully rely on knowledge learned by artificial intelligence. There is no guarantee that RL would learn deadlock detection completely and correctly.

8.3 Future Work

RL is still a very young and little researched area. There are many different methods in RL and finding the most suitable learning parameters is an additional challenge. A lot of effort is needed to achieve a high learning success by trial and error. During the time of this thesis, some first experiments with RL could be made. But based on the findings of this study, further research could be done for a long time. For example, the whole learning process could be performed with more complex test cases to deepen the learned knowledge. In addition, it could be investigated how the knowledge develops if the learning time is extended or other parameters are further modified.

In conclusion, RL is still a novel method, which could be applied in railway traffic. It is questionable whether RL at its current stage of development is already capable of taking on a major part in traffic management. However, this report revealed the potential of RL. Therefore, it should be considered to use RL at least as an additional support for existing traffic management.

9 References

Andreychuk, A., Yakovlev, K., Atzmon, D., & Stern, R. (2019). Multi-Agent Pathfinding with Continuous Time.

Barták, R., Švancara, J., & Vlk, M. (2018). A Scheduling-Based Approach to Multi-Agent Path Finding. *AAMAS*, (pp. 748-756). Stockholm.

Black, P. E. (2019, February 11). *Manhattan distance*. Retrieved January 17, 2020, from Dictionary of Algorithms and Data Structures: https://xlinux.nist.gov/dads/HTML/manhattanDistance.html

Cáp, M., Novák, P., Kleiner, A., & Selecký, M. (2014). *Prioritized Planning Algorithms for Trajectory Coordination of Multiple Mobile Robots.*

Coffman, E. G., Elphick, M. J., & Shoshani, A. (1971). System Deadlocks. *ACM Comput. Surv., 3*, 67–78.

Cohen, L., Wagner, G., Kumar, T. S., Choset, H., & Koenig, S. (2017). Rapid Randomized Restarts for Multi-Agent Path Finding Solvers.

Documentation: Algorithms: Shortest Paths. (2019). Retrieved January 9, 2020, from NetworkX: https://networkx.github.io/documentation/stable/reference/algorithms/shortest˙paths.html

Gawrilow, E., Klimm, M., Möhring, R. H., & Stenzel, B. (2012). Conflict-free vehicle routing: Load balancing and deadlock prevention. *EURO Journal on Transportation and Logistics*, 87-111.

Kelly, J. (2011). *How Sliding Puzzles Work*. Retrieved January 10, 2020, from HowStuffWorks: https://entertainment.howstuffworks.com/puzzles/sliding-puzzles.htm

Korte, B., & Vygen, J. (2018). *Combinatorial Optimization: Theory and Algorithms.* Berlin: Springer-Verlag.

Larman, C. (2005). *Applying UML and patterns : an introduction to object-oriented analysis and design and iterative development.* Prentice Hall PTR.

Li, J.-Q., Mirchandani, P. B., & Borenstein, D. (2007). The Vehicle Rescheduling Problem: Model and Algorithms. *Networks: An International Journal, 50*, 211-229.

Ma, H., Harabor, D., Stuckey, P. J., Li, J., & Koenig, S. (2018). *Searching with Consistent Prioritization for Multi-Agent Path Finding.*

Ma, H., Kumar, T. S., & Koenig, S. (2016). Multi-Agent Path Finding with Delay Probabilities.

Pachl, J. (2007). Avoiding Deadlocks in Synchronous Railway Simulations. *2nd International Seminar on Railway Operations Modeling and Analysis.* Hannover.

SBB Facts and Figures. (2019). Retrieved August 13, 2019, from SBB: https://reporting.sbb.ch

Schlechte, T. (2012). *Railway Track Allocation: Models and Algorithms.*

Sharon, G., Stern, R., Felner, A., & Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence 219*, 40-66.

Silver, D. (2005). *Cooperative Pathfinding.* American Association for Artificial Intelligence.

Standley, T. (2010). Finding Optimal Solutions to Cooperative Pathfinding Problems. *Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*, (pp. 173-178).

Standley, T., & Korf, R. (2011). Complete Algorithms for Cooperative Pathfinding Problems. *22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, (pp. 668-673). Barcelona, Catalonia, Spain.

We are SBB. (2019). Retrieved August 13, 2019, from SBB: https://company.sbb.ch/en

Zuo, X. (2009). An Immune Algorithm Based Robust Scheduling Methods. *Handbook of Research on Artificial Immune Systems and Natural Computing: Applying Complex Adaptive Technologies*, 124-140.