Routing Optimization for the Train Unit Shunting Problem in a Multi-Agent Deep Reinforcement Learning Framework

Jacob Trepat Borecka

Routing Optimization for the Train Unit Shunting Problem in a Multi-Agent Deep Reinforcement Learning Framework

by

Jacob Trepat Borecka

to obtain the degree of
Master of Science in Transport, Infrastructure & Logistics
at the Delft University of Technology,
to be defended publicly on Friday March 26, 2021 at 14:30

Student number:	4791428	
Project duration:	June 9, 2020 – March 26, 2021	
Committee:	Prof. Dr. R.M.P. Goverde	TU Delft
	Dr. N. Bešinović	TU Delft
	Dr. M.Y. Maknoon	TU Delft
	Dr. W. Lee	NS

Contents

Preface v

Executive Summary vii

List of Abbreviations xi

List of Figures xiii

List of Tables xvii

1 Introduction 1
- 1.1 Assignment Background 1
- 1.2 Objectives. 3
- 1.3 Research Questions 4
- 1.4 Thesis Outline 4

2 Theoretical Background 7
- 2.1 Markov Decision Process 7
- 2.2 Reinforcement Learning 8
- 2.3 Deep Reinforcement Learning 9
- 2.4 Reward Shaping. 10
- 2.5 Conclusions. 10

3 Literature Review 11
- 3.1 Railway Infrastructure Modelling and the Train Routing Problem 11
- 3.2 Shunting Yards and The Train Unit Shunting Problem 12
- 3.3 Reinforcement Learning and Deep Reinforcement Learning Applications in Operations Research Problems 17
- 3.4 Reinforcement Learning and Deep Reinforcement Learning Applications in the TUSP 20
- 3.5 Research Gap and Answer to Sub-Question 1 22

4 Problem Description 25
- 4.1 Infrastructure and Train Modelling 25
- 4.2 Train Unit Shunting Problem Subproblems 27
  - 4.2.1 Matching. 27
  - 4.2.2 Routing 28
  - 4.2.3 Parking. 29
  - 4.2.4 Service Tasks 29
  - 4.2.5 Splitting and Combining 30

5 Methodology 31
- 5.1 Overview 31
- 5.2 Agents 32
- 5.3 Environment 35
- 5.4 Neural Network and Algorithm for Policy Evaluation 37

6 Problem Approach 39
- 6.1 Heuristic for Random Exploration 40
- 6.2 Routing Strategies. 43
  - 6.2.1 Preliminar Observations on Rewards Shaping 43
  - 6.2.2 Type-Based Strategy 45
  - 6.2.3 In-Residence Time Strategy 48
  - 6.2.4 Overview of the Routing Strategies and Limitations of the Implementations 50
- 6.3 Answer to Sub-Question 2. 51

7 Adaptations and Formulation of Routing Strategies 53
7.1 Overview of Case Studies . . . 53
7.2 Adaptations of the Routing Strategies for Carrousel Shunting Yards without Flow Pattern Constraints . . . 55
7.3 Adaptations of the Routing Strategies for Carrousel Shunting Yards with Flow Pattern Constraints . . . 56
7.4 Adaptations of the Routing Strategies for Shuffleboard Shunting Yard Types . . . 57
7.5 Answer to Sub-Question 3. . . . 58

8 Experiments 61
8.1 Overview of Objectives and Experiments . . . 61
8.2 Experiments Setup . . . 62
8.2.1 Data Generation . . . 62
8.2.2 Objectives and Performance Metrics . . . 64
8.2.3 Benchmarking . . . 66
8.2.4 Learning Parameters . . . 66
8.2.5 Action Triggers . . . 67
8.2.6 Violations . . . 67
8.3 Experiment 1: Routing Strategies Performance Assessment with respect to Baseline Variant . . . 67
8.3.1 Setup. . . . 67
8.3.2 Results . . . 68
8.4 Experiment 2: Matching and Combination Problems with Single Rolling Stock Type in Carrousel Shunting Yard without Flow Pattern Constraints . . . 72
8.4.1 Setup. . . . 72
8.4.2 Results . . . 73
8.5 Experiment 3: Matching and Combination Problems with Multiple Rolling Stock Types in Carrousel Shunting Yard without Flow Pattern Constraints . . . 79
8.5.1 Setup. . . . 79
8.5.2 Results . . . 79
8.6 Experiment 4: Matching, Combination and Splitting Problems with Multiple Rolling Stock Types in Carrousel Shunting Yard with Flow Pattern Constraints . . . 83
8.6.1 Setup. . . . 83
8.6.2 Results . . . 84
8.7 Discussion . . . 88
8.8 Answer to Sub-Question 4. . . . 92

9 Conclusions and Recommendations 95
9.1 Conclusions. . . . 95
9.2 Recommendations . . . 99
9.2.1 Scientific Recommendations. . . . 99
9.2.2 Practical Recommendations . . . 100

Bibliography 101

A Scientific Paper 105

B Reward Functions 121

C List of Business Rules 131

Preface

This report is the result of my master thesis project, with which I conclude my studies in the Master of Science in Transport, Infrastructure & Logistics at the Delft University of Technology. It is therefore a moment of huge gratification. From June 2020 till March 2021, I worked on this thesis as an intern at the department of Research & Development Node Logistics at the Dutch National Railways NS. During this period, I have dived into the application of artificial intelligence techniques to solve a real-life problem in railway operations.

My deepest gratitude goes to Nikola Bešinović, with whom I have been working for over a year, for all the enriching discussions we have had during all the research we have done together, for the always very detailed feedback, and for caring about me, especially during the toughest times. To Wan-Jui Lee, for your patience and guidance towards artificial intelligence. It has been an exciting, steep learning curve. I hope this work is useful to you. To Yousef Maknoon, for all your support, trust and guidelines to frame my methodology. Lastly, to Rob Goverde, for your key feedback, especially notation-wise, and help to make my graduation possible. Thank you all for challenging me since the first day and helping me create this deliverable.

To Bob Huisman, for welcoming me in the department at NS, for making me feel part of the team and for some useful conversations about the department's reason to be and the context of the problem that I have been working on in particular. I highly appreciated your commitment to improving my internship experience despite the circumstances that forced us to work remotely during most of the project execution. I also want to thank Demian de Ruijter, Pim van den Bogaerdt, Joris den Ouden and Roel van den Broek from the department for their invaluable support, especially software-wise, and to the rest of my colleagues in the department for their inputs, as well as to my colleagues in our modest railway research group at TU Delft.

Thanks to my friends, including those from back home and especially those whom I had the chance to meet in Delft. To Raphael, for being always willing to listen to me. To Henar, I am so grateful I found you here in the very first days. To Patricia, you made things brighter with some remarkable stickers. And of course, to Javi, Arita, Maria del Mar, Júlia, Celia, Teresa. To Michumba, to Potluck, to Ruyterweg and to Boterbrug. I hope these words reflect all of them. You all have made me feel appreciated and supported. During my stay in Delft, I have realized that coming here was not just about getting a master's degree: it was mostly about people. This is what I will remember Delft for.

Finally, I must express my profound gratitude to my parents Santi (from whom I got my devotion for railways and transportation in general) and Ewa, and to my sister Marysia, for their unconditional love and support to all of my endeavours despite the distance that separates us.

To all of you, I sincerely thank you.

Enjoy reading!

Jacob Trepat Borecka
Delft, March 2021

Executive Summary

Problem Context

In busy passenger railway networks, large amount of trains have to be parked in shunting yards during night time. Shunting yards are sets of parallel tracks for sorting and storing trains off the mainline, so that the traffic on the mainline is not obstructed. For instance, the *Nederlandse Spoorwegen* (NS) is the largest passenger railway operator in the Netherlands, which manages a fleet of around 600 train units on 3,434 kilometers of rails, transporting up to 1.2 million passengers per day. In order to guarantee the high level of service expected by passengers, trains need to be cleaned, maintained and depart on time to start operating new train services. Train timetables are generally strongly linked with the operations in shunting yards, in the sense that the operating capacity of shunting yards is often the limiting factor towards dispatching a higher number of train services to the network, especially in busy railway networks. As a result, there is an increasing interest to study the train shunting processes and the operational capacity of shunting yards and how it can be maximized.

The associated activities in shunting yards, which include parking, cleaning, maintenance tasks, coupling and splitting of trains is known as the Train Unit Shunting Problem (TUSP). The TUSP has been traditionally solved by hand by railway planners, who apply human heuristics to solve it. However, this planning task is often difficult and time-consuming for humans. Also, routing strongly depends on the track layout in the yard (i.e. amount of parking tracks, connections between these and with service facilities, etc.). In practice, each yard is quite unique, but there are certain identifiable characteristics of the infrastructure layout that allow us to classify shunting yards in different types. Besides, routing is also influenced by the matching between arriving and departing trains, which can either be given or it needs to be solved otherwise.

With the gradual increase in passenger ridership, the commissioning of new trains is required to expand train fleets and to operate more services. Subsequently, the planning of shunting yard activities will become more and more challenging due to the increased railway traffic and the fact that in many networks the available infrastructure is reaching its capacity limits. One option to solve these infrastructure bottlenecks is to expand the infrastructure at shunting yards. Nevertheless, the commissioning of new infrastructure are costly and physical room for expansion is often very limited. Consequently, in order to increase the infrastructure capacity, it is essential to explore how to optimize the usage of the already existing infrastructure.

Artificial intelligence approaches have been found to be powerful to solve some challenging problems in the field of operations research the past few years. One of the motivations to use Deep Reinforcement Learning (DRL) in the TUSP is the consistency of its outputs for similar problems, which is appreciated by planners for the acceptance of Planning Assistance Tools. Planning of tasks at the operational and real-time planning level such as train shunting are performed on a daily basis, for which solution algorithms must be very fast. This is possible by using DRL, since a DRL-based model can be trained in advance and for its use it simply has to perform inference of results in a specific real-life problem. Some promising developments of DRL for TUSP can be found in literature.

Goal and Scope of Research

This paper develops an heuristic for random exploration and two routing strategies in a multi-agent DRL framework for the TUSP. This approach addresses different problem designs, such as different shunting yard types and different matching problems, and to contribute towards a better learning process. The developments help agents make the right decisions from the logistics efficiency point of view and consequently to produce better route plans. The work extends on previous research. This is hence the focus of our research, which constitutes a novel approach from the scientific point of view to implement routing within the TUSP, in a multi-agent DRL framework, especially since the application of artificial intelligence in railway operations are relatively scarce. The main contributions of this research can be summarized as follows:

- An heuristic for random exploration in a multi-agent DRL framework for the TUSP based on the principles of large neighbourhood search.
- Formulation of two routing strategies for the TUSP in a multi-agent DRL framework by means of reward functions.
- Two real-life case studies in the Dutch railway network.
- An analysis of the flexibility of routing strategies to be applied on different types of shunting yards and different matching problems.

The main research question is formulated as follows: *How can routing strategies for the Train Unit Shunting Problem in a Multi-Agent Deep Reinforcement Learning framework help optimizing route plans to support railway planners?*

Research Methodology

The framework developed by NS upon which this research builds on is based on two models. First, the Train Maintenance and Shunting Simulator (TORS), which is the simulation environment for sequential planning. Second, the Multi-Agent Train Unit Deep Reinforcement Learning (MATDRL), which is a programme that uses DRL to solve planning problems for shunting yards. It does so by treating each train unit as a separate agent, which chooses the action for its train unit using the value iteration algorithm. Figure 1 shows a high level representation of the framework described, including the inputs to TORS and MATDRL.

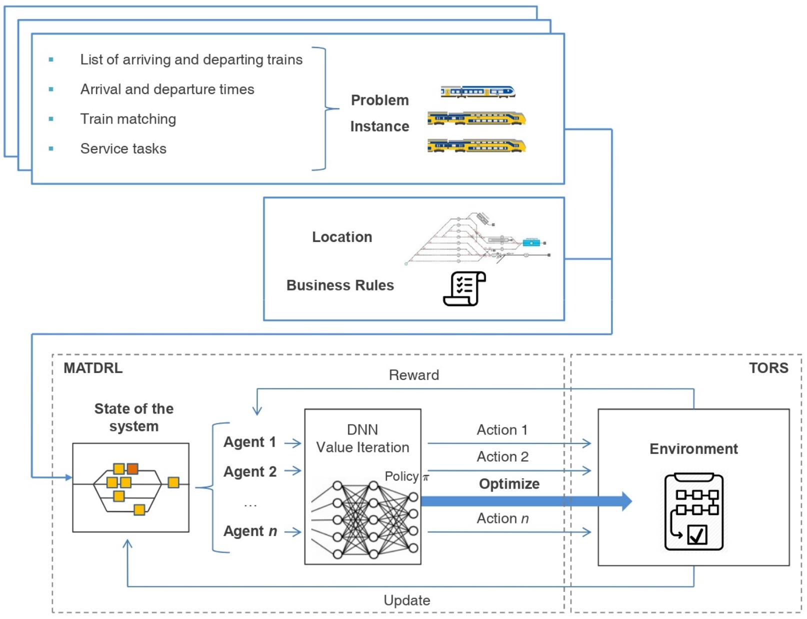

Figure 1: High-level representation of the DRL framework for the TUSP.

We developed an heuristic for random exploration and two routing strategies in the multi-agent DRL framework for the TUSP developed by NS. Each routing strategy consists of four components: (1) *standard parking rules*, (2) *combination and split rules*, (3) *conflict resolution rules* and (4) *unnecessary movements rules*. First, we develop the **Type-Based Routing Strategy (TBS-RS)**, which is based on the rule of grouping trains of the same rolling stock type on the same tracks (Figure 2). Second, the **In-Residence Time Routing Strategy (IRTS-RS)**, which is based on the rule of assigning train units to parking tracks based on its departure time, creating a chronological order of trains on each track (Figure 3).

We performed 4 experiments. First, an experiment to assess the performance of the routing strategies with respect to a baseline variant. Second, two experiments with matching and combination problems in a *carrousel* shunting yard without flow pattern constraints. Lastly, an experiment with matching, combination

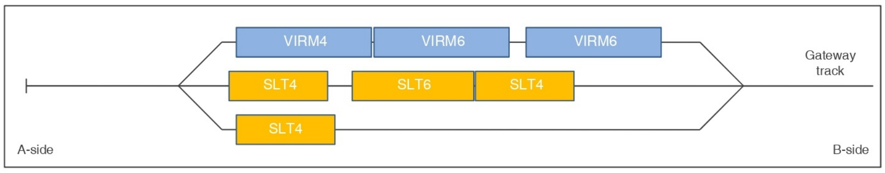

Figure 2: Example of the TBS parking rules with trains of types VIRM and SLT.

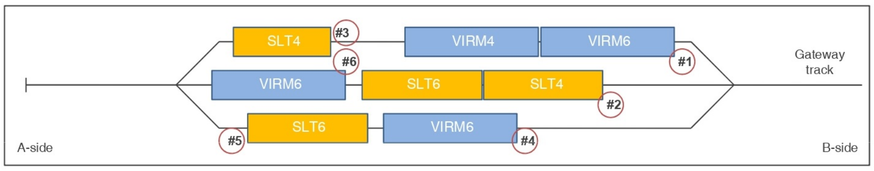

Figure 3: Example of the IRTS parking rules with trains of types VIRM and SLT. The position of each train in the departure sequence is shown next to each train.

and splitting in a *carrousel* shunting yard with flow pattern constraints. The results are benchmarked against HIP, which is currently the best performing programme to solve the TUSP. As an example, Figure 4 visualizes the solution produced for one particular instance by TORS-MATDRL with the IRTS-RS. The figure represents the task schedules for each train unit in the problem, indicated by their corresponding ID on the vertical axis, and ordered from top to bottom according to their arrival times. Time is represented on the horizontal axis.

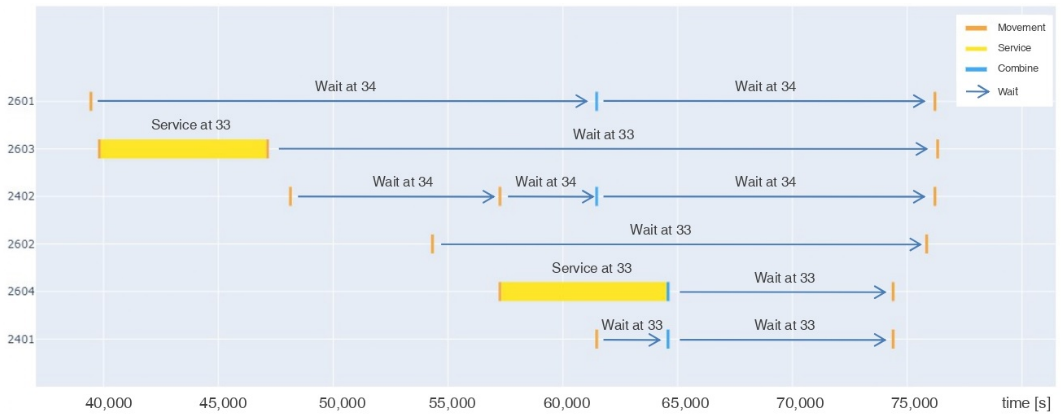

Figure 4: Task schedule representing the solution to one particular instance in Heerlen.

Conclusions

In this research we have developed an heuristic for random exploration and two routing strategies for the Train Unit Shunting Problem in a multi-agent DRL framework adapted for two different case studies. The application of the implementations in two real-life case studies in the Dutch railway network has demonstrated its potential to produce more efficient route plans and to adapt to different problems designs such as different matching problems types and different shunting yards. For problems with 6 train units, the routing strategies have increased the solvability from less than 10% to up to 55% and reduce the total number of movements by 25%. The average number of movements per train in the case study of Heerlen is about 2.1-2.6, whereas in Kleine Binckhorst it ranges between 4.4 and 4.9. Problems with 100% matching seem to converge faster than problems with 0% matching, which are more relaxed. Shunting yards with flow pattern constraints also seem to be more constrained than those without such constraints, and agents tend to create a bottleneck on the relocation track, which they find difficult to resolve. The solvability achieved in this case study is up to 15% compared to 30-55% in the experiments case study without flow constraints. In summary, our results demonstrate that DRL has potential to solve routing optimization problems efficiently and effectively.

List of Abbreviations

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
ATB	*Automatische Treinbeïnvloeding* (Dutch automatic train protection system)
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
EMU	Electric Multiple Unit
FCFS	First Come First Served
HIP	Hybrid Integrated Planning
HRE	Heuristic for Random Exploration
IRTS	In-Residence Time Strategy
KPI	Key Performance Indicator
LIFO	Last In First Out
MATDRL	Multi-Agent Train Unit Deep Reinforcement Learning
MDP	Markov Decision Process
MILP	Mixed-Integer Linear Programming
MIP	Mixed-Integer Programming
NS	*Nederlandse Spoorwegen* (Dutch Railways)
OPG	*Opstel Plan Generator* (Stabling Plan Generator)
OR	Operations Research
RCP	Robustness in Critical Points
RL	Reinforcement Learning
RS	Routing Strategy
SBB	*Schweizerische Bundesbahnen* (Swiss Federal Railways)
SL	Service Location
SLT	*Sprinter Lighttrain*
TBS	Type-Based Strategy
TORS	*Trein Onderhoud en Rangeersimulator* (Train Maintenance and Shunting Simulator)
TRP	Train Routing Problem
TSP	Travelling Salesman Problem
TUSP	Train Unit Shunting Problem
VIRM	*Verlengd InterRegio Materieel* (lengthened interregional rolling stock)
VRP	Vehicle Routing Problem
VSP	Vehicle Scheduling Problem

List of Figures

1 High-level representation of the DRL framework for the TUSP. viii
2 Example of the TBS parking rules with trains of types VIRM and SLT. ix
3 Example of the IRTS parking rules with trains of types VIRM and SLT. The position of each train in the departure sequence is shown next to each train. ix
4 Task schedule representing the solution to one particular instance in Heerlen. ix

1.1 Definitions of different parts of a train (Goverde, 2018). 1
1.2 Flow associated to passenger train shunting at a node. 2

2.1 Basic structures of the three learning paradigms: supervised learning, reinforcement learning, and unsupervised learning. Source: Wang et al. (2012). 8
2.2 Representation of the interaction between agent and environment in Deep Reinforcement Learning. Source: Mao et al. (2016). 10

3.1 Kleine Binckhorst shunting yard (Sporenplan Online, 2019). The entry tracks are marked with blue circles. 12
3.2 Cartesuisweg shunting yard (Sporenplan Online, 2019). The entry tracks are marked with blue circles. 13
3.3 Layout of the node of Eindhoven, with the station area (1) and two shunting yards: the Garden (2) and the East-side (3) (Sporenplan Online, 2019). 14
3.4 Example of an activity graph in the local search framework for the TUSP. Source: Van Den Broek (2016). 15
3.5 Visualization of a simple Flatland environment with one train station and one train (in red). Source: Mohanty et al. (2020). 20

4.1 Representation of track sides. 25
4.2 Kleine Binckhorst shunting yard (Sporenplan Online, 2019). The entry tracks are marked with blue circles. 26
4.3 Representation of feasible matchings for a particular instance. 28
4.4 Representation of one possible solution to a matching problem. 28
4.5 Representation of a saw movement. 28

5.1 High-level representation of the multi-agent DRL framework for the TUSP. 32
5.2 Actions representation. 34
5.3 Example of the initial part of a decision tree with triggers and actions going on in time in a shunting yard for one train unit. Train units arrive at track 84. We assume that all setback and moving actions take 120 seconds and therefore the next trigger takes place 120 seconds after such action can be taken. 37
5.4 Simplified representation of the shunting yard in Heerlen. 37
5.5 Example of an instance solved with 3 train units in Heerlen. Notes: track 84 is the gateway track and setback time is set to zero. 38

6.1 Extensions implemented on top of TORS-MATDRL. 39
6.2 Conceptual model of the development of implementations on top of TORS-MATDRL. 40
6.3 Representation of the conventions used for routing strategies. 44
6.4 Example of the TBS standard parking rules with trains of types VIRM and SLT. 45
6.5 Examples of conflicting scenarios to illustrate the preconditions for conflict resolution in the TBS-based routing strategy. 47

6.6 Representation of the resolution of a conflict by means of relocation. The train unit VIRM4 at the back of the queue is the only possible candidate for the next departure. It moves to the relocation track (1) and then to the gateway track via a non-conflicting path (2). 48
6.7 Example of the IRTS standard parking rules with trains of types VIRM and SLT. The position of each train in the departure sequence is shown next to each train. 48
6.8 Representation of a state that creates a conflicting departure sequence (1) and hence it creates the need for relocation, and how relocation can solve the conflict (2-3). 50

7.1 Heerlen shunting yard (top centre, dashed lines consisting of tracks 31 to 39) and station (right) (Sporenplan Online, 2019). 55
7.2 Kleine Binckhorst shunting yard (Sporenplan Online, 2019). The entry tracks are marked with blue circles. 55
7.3 Kleine Binckhorst shunting yard with operation details (adapted from Sporenplan Online, 2019). 55
7.4 Cartesuisweg shunting yard (Sporenplan Online, 2019). The entry tracks are marked with blue circles. 58

8.1 Train unit types and subtypes used in the experiments with details. Drawings from Treinposities (2021). 64
8.2 Experiment 1 0% matching no RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 70
8.3 Experiment 1 0% matching no RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 70
8.4 Experiment 1 0% matching no RS average solved during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 70
8.5 Experiment 1 0% matching no RS total reward during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 70
8.6 Experiment 1 0% matching TBS-RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 70
8.7 Experiment 1 0% matching TBS-RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 70
8.8 Experiment 1 0% matching TBS-RS average solved during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 70
8.9 Experiment 1 0% matching TBS-RS total reward during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 70
8.10 Experiment 1 100% matching no RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 71
8.11 Experiment 1 100% matching no RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 71
8.12 Experiment 1 100% matching no RS average solved during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 71
8.13 Experiment 1 100% matching no RS total reward during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 71
8.14 Experiment 1 100% matching IRTS-RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 71
8.15 Experiment 1 100% matching IRTS-RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 71
8.16 Experiment 1 100% matching IRTS-RS average solved during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 71
8.17 Experiment 1 100% matching IRTS-RS total reward during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 71
8.18 Experiment 2 TBS-RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 74
8.19 Experiment 2 TBS-RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 74
8.20 Experiment 2 IRTS-RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 75

8.21 Experiment 2 IRTS-RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 75
8.22 Task schedule representing the solution to one particular instance in Heerlen by TORS-MATDRL with the IRTS-RS. 78
8.23 Example of the detailed route of train unit 2603 in one particular instance in Heerlen by TORS-MATDRL with the IRTS-RS. Adapted from Sporenplan Online (2019). 78
8.24 Experiment 3 TBS-RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 80
8.25 Experiment 3 TBS-RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 80
8.26 Experiment 3 IRTS-RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 81
8.27 Experiment 3 IRTS-RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 81
8.28 Experiment 4 TBS-RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 84
8.29 Experiment 4 TBS-RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 84
8.30 Experiment 4 IRTS-RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 85
8.31 Experiment 4 IRTS-RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6. 85
8.32 Task schedule representing the solution to one particular instance from Experiment 4 in Kleine Binckhorst. 87
8.33 Example of the detailed route of train unit 2401 in one particular instance in Kleine Binckhorst by TORS-MATDRL with the TBS-RS. Adapted from Sporenplan Online (2019). 87

List of Tables

1.1 Report outline. 5

3.1 Overview of some recent research on RL and DRL in the operations research domain. 21
3.2 Overview of recent research on the TUSP with DRL. M: matching; S: servicing; P: parking; R: routing; S&C: splitting and combining; SA: single agent; MA: multi-agent; DQN: Deep Q-Network. 22

4.1 Infrastructure notation used for the TUSP problem. 25
4.2 Overview of the TUSP subproblems. 27

5.1 List of rewards and their values in TORS-MATDRL. 36
5.2 List of arrival and departure events for an example instance. Time is given in seconds. 37

6.1 List of rewards and their values. 45
6.2 Overview of the TBS and IRTS-based routing strategies. 51

7.1 Overview of differences and commonalities between the shunting yards of Heerlen and Kleine Binckhorst. 54

8.1 Overview of service tasks details used in the case studies in Heerlen. 63
8.2 Duration of events. 63
8.3 List of key performance indicators used. Averages are defined per train unit. 66
8.4 Main learning parameters for experiments. 67
8.5 List of arriving and departing trains instances for Experiment 1 (unordered). 68
8.6 Middle triggers added in Experiment 1. 68
8.7 Learning parameters in Experiment 1. 68
8.8 Computational results of the training process in Experiment 1. 72
8.9 Performance metrics for Experiment 1. 73
8.10 List of arriving and departing trains instances for Experiment 2 (unordered). 73
8.11 Middle triggers added in TORS-MATDRL for Experiment 2. 74
8.12 Learning parameters in Experiment 2. 74
8.13 Computational results of the training process in Experiment 2. 74
8.14 Performance metrics for Experiment 2. 75
8.15 List of arriving and departing trains instances for Experiment 3 (unordered). 79
8.16 Middle triggers added in TORS-MATDRL for Experiment 3. 79
8.17 Learning parameters in Experiment 3. 79
8.18 Computational results of the training process in Experiment 3. 80
8.19 Performance metrics for Experiment 3. 82
8.20 List of arriving and departing trains instances for Experiment 4 (unordered). 83
8.21 Middle triggers added in Experiment 4. 83
8.22 Learning parameters in Experiment 4. 84
8.23 Computational results of the training process in Experiment 4. 84
8.24 Performance metrics for Experiment 4. 88

List of Algorithms

1 Heuristics for Assignment of Rewards 36

2 Heuristic for Random Exploration in TORS-MATDRL Pseudocode 42

3 Heuristics for Assignment of Rewards with TBS-Based Routing Strategy in Heerlen Pseudocode in TORS-MATDRL (Part 1) 122

3 Heuristics for Assignment of Rewards with TBS-Based Routing Strategy in Heerlen Pseudocode in TORS-MATDRL (Part 2) 123

4 Heuristics for Assignment of Rewards with IRTS-Based Routing Strategy in Heerlen Pseudocode in TORS-MATDRL (Part 1) 124

4 Heuristics for Assignment of Rewards with IRTS-Based Routing Strategy in Heerlen Pseudocode in TORS-MATDRL (Part 2) 125

4 Heuristics for Assignment of Rewards with IRTS-Based Routing Strategy in Heerlen Pseudocode in TORS-MATDRL (Part 3) 126

5 Heuristics for Assignment of Rewards with TBS-Based Routing Strategy in Kleine Binckhorst Pseudocode in TORS-MATDRL (Part 1) 127

5 Heuristics for Assignment of Rewards with TBS-Based Routing Strategy in Kleine Binckhorst Pseudocode in TORS-MATDRL (Part 2) 128

6 Heuristics for Assignment of Rewards with IRTS-Based Routing Strategy in Kleine Binckhorst Pseudocode in TORS-MATDRL (Part 1) 129

6 Heuristics for Assignment of Rewards with IRTS-Based Routing Strategy in Kleine Binckhorst Pseudocode in TORS-MATDRL (Part 2) 130

1 Introduction

1.1. Assignment Background

The Netherlands currently has the second busiest rail passenger transport system in Europe after Switzerland, with a daily ridership above 1.2 million passengers (NS, 2016) on 3,434 kilometres of network length (ProRail, 2016). Its main passenger operator, known in Dutch as the *Nederlandse Spoorwegen*, manages a fleet of around 600 train units, which operate around 4,800 daily train services. During peak hours, most of the rolling stock is used to operate the scheduled train services with sufficient capacity to absorb the high passenger demand. Nevertheless, during off-peak hours and especially during night time, the operator has to park a large amount of train units in shunting yards. Shunting yards are in essence complex sets of parallel tracks for sorting and storing trains off the mainline so that the traffic on the mainline is not obstructed. These are usually located next to stations where trains start and end their service routes. In order to guarantee the level of service expected by passengers, trains need to be cleaned, maintained and to depart on time to start operating new train services. Train timetables are generally strongly linked with the operations in shunting yards, in the sense that the operating capacity of shunting yards is often the limiting factor towards dispatching a higher number of train services to the network, especially in busy railway networks. As a result, there is an increasing interest to study the train shunting processes and the operational capacity of shunting yards and how it can be maximized.

The associated activities in shunting yards, which include parking, cleaning and maintenance tasks, is what we understand as the train shunting processes, which is planned beforehand. On the other hand, the physical complex consisting of interconnected stations and shunting yards is what we understand as a node (ProRail, 2019). Also, we define a train unit, as a generalization of Electric Multiple Unit (EMU), as a composition of one or multiple carriages in a fixed order that are self-propelled. A train is a composition of one or more carriages, which can consist at the same time of one or multiple train units to provide passenger service (Figure 1.1).

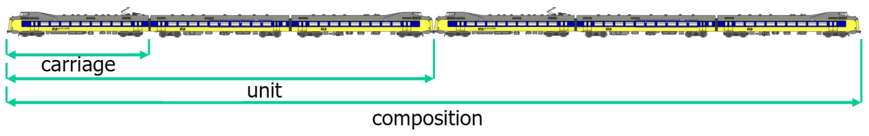

Figure 1.1: Definitions of different parts of a train (Goverde, 2018).

The planning of activities at a node includes all the processes that need to be carried out for each train since the very last passenger alights the train at its destination and then the train moves from the platform to a shunting yard, until the same train goes back to a passenger platform and starts a new service from the same node. During this time, some cleaning, inspection and maintenance activities need to be carried out to ensure the rolling stock readiness for the follow-up train services at a certain level of service expected by passengers. In particular, the tasks included in node planning are parking of trains, mechanical checks, interior cleaning of trains, exterior cleaning of trains, minor reparations, shunting and coupling and splitting of train compositions.

The core of the planning of such activities at nodes is the Train Unit Shunting Problem (TUSP). The TUSP is the planning problem concerning the assignment of train units from shunting yards to scheduled train services in such a way that the resulting operations are non-conflicting with each other (Haahr et al., 2017). At the high-level, the problem involves matching train units from arriving to departing train services, but in order to accomplish so, it also involves the assignment of these trains to appropriate routes and shunting yard tracks for parking, and splitting and combining them as well as servicing them. Figure 1.2 shows a high-level representation of the flow associated to train shunting at a node.

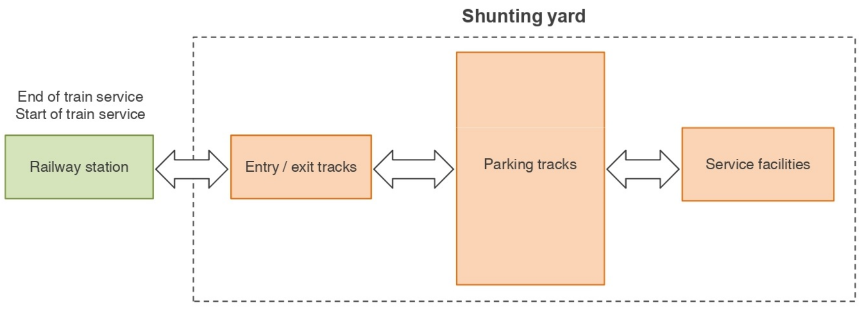

Figure 1.2: Flow associated to passenger train shunting at a node.

The TUSP has been traditionally solved by hand by railway planners at NS, who apply human heuristics to solve its different subproblems, such as parking strategies to find routes for train shunting movements. A train route is defined as an infrastructure itinerary assigned to each train over a specific time period, and a route plan represents a set of compatible train routes. However, this planning task is not trivial since planners can often decide among many different possibilities for each train shunting movement and thus the number of possible states that have to be evaluated becomes quite large even for simple instances. Therefore, solving such problems by hand is difficult and time-consuming.

In the Netherlands, NS expects an increase in passenger ridership in the upcoming years, which requires the commissioning of new trains to expand its fleet. Subsequently, the planning of node activities will become more and more challenging due to the increased railway traffic and the fact that the available infrastructure is reaching its capacity limits. One option to solve these infrastructure bottlenecks is to expand the infrastructure at shunting yards. Nevertheless, the commissioning of new infrastructure is costly and the physical room for expansion is often very limited. Consequently, in order to increase infrastructure capacity, it is essential to explore how to optimize the usage of the already existing infrastructure, which has already been a focus of research for NS over the last few years. In general, exploiting the capacity of the railway infrastructure is one of the most challenging aspects of railway planning in all railway networks.

The department of Research & Development Node Logistics at NS is researching the capacity of nodes and is currently devoted to developing software approaches as Planning Assistance Tools to solve the TUSP that could help automatizing the planning of activities at nodes and to optimize the associated operations. These approaches include classic optimization techniques, local search heuristics and artificial intelligence, based on deep reinforcement learning (DRL).

Artificial intelligence approaches have been found to be powerful to solve some challenging problems in the field of operations research the past few years. However, AI applications in railway operations are still relatively scarce. One of the main motivations to use DRL in the TUSP problem is the consistency of its outputs for similar problems, which was proved by Peer et al. (2018), which is appreciated by planners for the acceptance of Planning Assistance Tools. This feature of DRL is not necessarily the case for classic optimization approaches and local search heuristics (Peer et al., 2018). By contrast, with RL it is easier to understand the process of inference, it delivers more consistent results and it is easier to communicate an exact solution. Finally, it should be noted that tasks at the tactical planning level such as timetabling are performed with a periodicity of a year and thus, in these cases, classic optimization approaches, which are executed from scratch, might still convenient. However, tasks at the operational and real-time planning level such as train shunting are performed on a daily basis. In these cases, solution algorithms must be very fast. This is possible by using DRL, since a DRL-based model can be trained in advance and for its use, it simply has to perform inference of results in a specific real-life problem.

The Dutch railways have researched deep reinforcement learning (DRL) techniques to approach the TUSP with promising results (Peer et al., 2018; Jamshidi, 2019; Cohen, 2019; Kyziridis, 2019; Barnhoorn, 2020). All the work carried out so far has contributed towards the current model (NS, 2020), based on a multi-agent system, in which most of the effort has been put on the computational aspects of the DRL framework. After Beerthuizen (2018) transferred ideas and policies from container handling problems to the TUSP and applied them in a discrete event simulation framework with promising results, NS is willing to similarly incorporate knowledge and aspects from the logistics side of these problems to the multi-agent DRL framework. The goal is to contribute towards a better learning process of the neural network, to help the model to make the right decisions from the logistics efficiency point of view and consequently to produce better route plans.

One of the key elements of the TUSP that NS is willing to formulate in the DRL framework is routing. Routing is in practice an overarching part of the TUSP since it comprises all the movements from the arrival of the train at the shunting yard until its departure, required to carry out service tasks, parking, splitting and combining trains, which often take place at different location in the shunting yard, for each train unit. Routing must guarantee that trains depart on time, with its service tasks completed and in the right composition according to the schedule. This sequence of decisions has to respect the physical constraints inherent of railway operations, i.e. non-conflicting routes and avoiding trains blocking each other. Therefore, agents need more guidance towards learning to do the right movements.

It is important to note that, while activities such as parking, servicing, splitting or combining are very standardized in the sense that they are carried out at any shunting yard using the same procedures, route plans strongly depend on the track layout in the shunting yard. In practice, each shunting yard is pretty much unique, but there are certain identifiable characteristics of the infrastructure layout that allow us to classify shunting yards in different types. For instance, tracks can be either accessible from one single side (dead-ended tracks, or last-in-first-out (LIFO) tracks) or from both sides. As a result, the configuration of the shunting yard, i.e. the amount of parking tracks and its types, and the connections between those, as well as the location of service facilities define the possibilities to move trains.

There are however additional factors to take into account in routing. Routing should be scalable in a way that it can handle a certain amount of trains such that makes it applicable for real-life cases. Also, at NS, in practice, for the assessment of the capacity of shunting yards, train departures only consist of a list of requested rolling stock types and subtypes. By contrast, for planning of daily operations a detailed list of departing train units is given. This has an effect on the task of matching arriving train units to departing trains units in the TUSP, which either has to be solved in the former case or the matching is already given and hence solved in the latter case. This is a major contributing factor to how parking, combination and splitting of trains can be solved.

Therefore, it is necessary to develop conceptual routing strategies such that take all the above mentioned factors, i.e. mainly the shunting yard type, its scalability and train matching are taken into account, and to formulate them in the DRL framework by NS (2020). This is hence the focus of our research, which constitutes a novel approach from the scientific point of view to implement routing within the TUSP, in a multi-agent deep reinforcement learning framework, especially since the application of artificial intelligence in railway operations are relatively scarce. From the learning point of view, more guidance to agents will inevitably result in a more constrained search space, which can significantly help speeding up learning, although there is a higher risk to fall into local minima.

1.2. Objectives

This paper develops an heuristic for random exploration and two routing strategies in a multi-agent DRL framework for the TUSP. This approach addresses different problem designs, such as different shunting yard types and different matching problems, and to contribute towards a better learning process. The developments help agents make the right decisions from the logistics efficiency point of view and consequently to produce better route plans. The work extends on previous research (Barnhoorn, 2020; NS, 2020). This is hence the focus of our research, which constitutes a novel approach from the scientific point of view to implement routing within the TUSP, in a multi-agent DRL framework, especially since the application of artificial intelligence in railway operations are relatively scarce.

Routing strategies strongly depend on factors such as the infrastructure layout on shunting yards as well as on train matching. Consequently, adaptations are needed to generic routing strategies in order to fit particular problems. Therefore, we assess how flexible are the routing strategies developed in different contexts, such as different shunting yard types and train matching problems. The goal is to gain insights from both the railway operations point of view, and our multi-agent DRL framework point of view. With respect to railway operations, we are interested to see how generic can the routing strategies be, what is shared in common and what differences present in different contexts, and the reasons behind that. Besides, with respect to the multi-agent DRL framework, we are also interested to see what is the scalability and limitations of the routing strategies so that we can establish a solid direction towards further developments and perhaps even to solve other routing problems using a multi-agent DRL framework.

Therefore, the main contributions of this research can be summarized as follows:

- An heuristic for random exploration in a multi-agent DRL framework for the TUSP based on the principles of large neighbourhood search.
- Formulation of two routing strategies for the TUSP in a multi-agent DRL framework by means of reward functions.
- Two real-life case studies in the Dutch railway network.
- An analysis of the flexibility of routing strategies to be applied to different types of shunting yards and matching problems.

1.3. Research Questions

Taking into account all the previous considerations, the main research question can be formulated as follows:

How can routing strategies for the Train Unit Shunting Problem in a Multi-Agent Deep Reinforcement Learning framework help optimizing route plans to support railway planners?

This can be further detailed by specifying the research sub questions stemming from the main question:

- SQ1. What information from the logistics side of routing in the Train Unit Shunting Problem can be incorporated into the current multi-agent deep reinforcement learning framework developed by NS?
- SQ2. How can routing strategies be formulated in a multi-agent deep reinforcement learning framework in order to produce optimal route plans and to speed up learning?
- SQ3. How flexible and scalable are the routing strategies in different shunting yards?
- SQ4. What is the performance of the new framework?

1.4. Thesis Outline

The remainder of this report is organized as follows: Chapter 2 introduces some relevant theoretical background on artificial intelligence and deep reinforcement learning in particular. Chapter 3 discusses relevant literature on the Train Unit Shunting Problem, applications of reinforcement learning in operations research problems and the research gap. Chapter 4 describes in detail the Train Unit Shunting Problem and its subproblems. Chapter 5 describes the methodology used in this work, based on a multi-agent deep reinforcement learning framework. Chapter 6 presents the extensions developed in this research as a routing optimization solution. Chapter 7 presents the adaptations of the routing strategies for specific shunting yards types and its formulation in the TORS-MATDRL framework. Chapter 8 presents the experiments carried out and the discussion of the results. Finally, Chapter 9 presents the conclusions and recommendations of this research. Table 1.1 summarizes the thesis outline described.

Chapter	Title	Research Sub Question
2	Theoretical Background	-
3	Literature Review	1
4	Problem Description	-
5	Methodology	-
6	Problem Approach	2
7	Adaptations and Formulation of Routing Strategies	3
8	Experiments	4
9	Conclusions and Recommendations	-

Table 1.1: Report outline.

2 Theoretical Background

In this chapter we introduce some theoretical background relevant for this research. Artificial intelligence techniques have been found to be helpful to solve many challenging problems in computer science and operations research. In order to understand the principles of Deep Reinforcement Learning (DRL), we focus first on its associated core concepts, which include the Markov Decision Processes and Dynamic Programming. Subsequently, we review the Reinforcement Learning area of Machine Learning and we discuss the well-known Q-learning algorithm. Finally, we give a brief overview of Deep Learning and we focus on Deep Reinforcement Learning and the principles of reward shaping.

2.1. Markov Decision Process

Markov Decision Processes (MDP) are a discrete time stochastic control process. In other words, MDPs are a type of probabilistic sequential decision model. The main idea of MDPs is that, at each time step, the process is in a state s, from which an agent or decision maker may choose an action a from a set of allowed actions using a policy, and transition to a new state s' and receive a reward for it. Besides, the state transitions of an MDP satisfy the Markov property, i.e. the memoryless property of stochastic processes, which states that the future states depend only upon the current state, and not on the sequence of events that preceded it. If we assume that the set of all possible states is finite, and the set of all possible actions in each state as well, then we have a finite MDP.

MDPs are in essence a generalization of the Markov chains that include multiple possible actions and rewards (Puterman, 2005). In Markov chains, by contrast, there are no rewards, although each state can still be reached with a certain probability, which can also be equal to zero. Finite MDPs are useful for studying optimization problems solved via dynamic programming and reinforcement learning.

An MDPs is a 4-tuple (S, A, P, R) where:

- S is a finite set of states.
- A is a finite set of actions.
- $P : S \times A \times S \to [0,1]$; $P(s,a,s') = P(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability that an action a in state s at time t leads to a state s' at time $t+1$.
- $R : S \times A \times S \to R$ is a scalar reward function; $R(s,a,s')$ is the immediate reward received after transitioning from state s to state s', due to action a.

The core problem of the MDPs is to find a policy for the agent, i.e. a function π that specifies the action $\pi(s)$ that the agent will take when in a given state. Once the policy is defined, the action that the agent will take is fixed, and as a result the process boils down to a Markov chain. As a result, a separate Markov chain is associated to each policy. The goal is to define a policy such that it maximizes expected rewards over time (Powell, 2019). Bellman's optimality equation writes the value of being in a discrete state s at time step t as

$$V_t(s) = \max_{a \in A_s} \left(r(s,a) + \sum_{s' \in S} P\left(s' \mid s,a\right) V_{t+1}\left(s'\right) \right), \tag{2.1}$$

according to Puterman (2005), where $P(s'|s,a)$ is the one-step transition matrix defined above.

2.2. Reinforcement Learning

Machine learning, as a subset of artificial intelligence, is the study of computer algorithms that improve automatically through experience. In essence, machine learning algorithms have the ability to learn without being explicitly programmed. There are three types of learning: *supervised*, *unsupervised* and *reinforcement learning*. Supervised learning is task-driven, which learns a function that maps an input to an output based on example input-output pairs. Thus, the job is to replicate the right answer. By contrast, unsupervised learning is data-driven, where machine learning looks for previously undetected patterns (i.e. identify clusters) in a data set with no preexisting labels and with a minimum of human supervision. Reinforcement learning is an area of machine learning where agents take random actions to explore and interact with a dynamic environment, so that they may then learn behaviour and determine which actions lead them to the best payoffs (Kaelbling et al., 1996). Figure 2.1 shows the basic structures of the three machine learning paradigms.

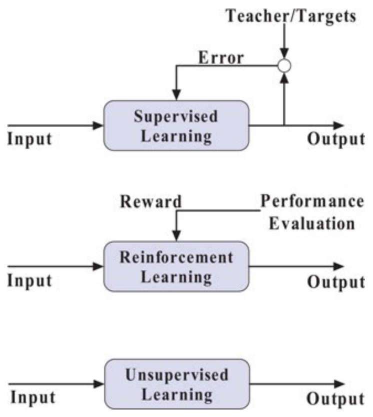

Figure 2.1: Basic structures of the three learning paradigms: supervised learning, reinforcement learning, and unsupervised learning. Source: Wang et al. (2012).

In reinforcement learning, agents are given a description of the current state and have to choose the next action from a set of possible actions so as to maximize a scalar feedback received after each action (Sharpanskykh, 2019). The agent's environment can be modeled by a discrete time, finite state Markov decision process. Each agent maintains a policy that maps the current state into the desirable action(s) to be performed in that state. The value of policy π at state s, $V^{\pi}(s)$, is defined by:

$$V^{\pi}(s) \overset{\text{def}}{=} E\{\sum_{t=0}^{\infty} \gamma^t r^{\pi}_{s,t}\}, \tag{2.2}$$

where $r^{\pi}_{s,t}$ is the random variable corresponding to the reward received by the learning agent t time steps after it starts using the policy π in state s, and γ is a discount rate ($0 < \gamma < 1$). The discount factor concerns the relative importance of rewards that are received later in the sequence with respect to the rewards obtained at the present moment. If the discount factor is very low, the focus of the agents will be on immediate rewards, whereas a γ closer to 1 will make the agents more far-sighted and thus take future rewards strongly into account.

RL shows to be powerful in large-scale and complex problems of dynamic optimization, since RL is able to deal with the problems stemming from the curse of dimensionality and the curse of modelling of classic dynamic programming, which makes the latter ineffective for large instances (Gosavi, 2009). We understand the phenomena of the curse of dimensionality as the challenge of problems with a large dimension, and the phenomena of the curse of modelling as the difficulty to compute the values of transition probabilities in complex systems.

Instead of learning the value $V(s)$ of being in a state s, the core algorithmic strategy of reinforcement learning involves learning the value $Q(s,a)$ of being in a state s and then taking an action a (Powell, 2019). The basic algorithm is known as Q-learning, which is a model-free reinforcement learning algorithm which allows finding a specific policy π^* such that maximizes the value of the policy $V^{\pi}(s)$ for all states $s \in S$. The decision policy is represented by a function $Q : S \times A \to R$, which represents the long-term discounted rewards for each state-action pair, and action a to perform in a state s is chosen such that it maximizes the reward:

$$V(s) = \max_{a} Q(s,a) \quad \forall s \in S \tag{2.3}$$

A relevant difficulty of RL is the dilemma of explorations versus exploitation. This raises the question on when the agent should try out (perceived) non-optimal actions in order to explore the environment (and potentially improve the model) and when should it exploit the optimal action in order to make useful progress (Arulkumaran et al., 2017), simulating a human interacting with an unknown environment. This dilemma is actually a challenge in reinforcement learning since it has a strong impact on the results. In Q-learning and other model-free RL algorithms it is typical to use a simple ε-greedy exploration policy in which, for exploration, the agent keeps trying all the actions in all the states with nonzero probability: a random action is executed with probability $\varepsilon \in [0,1]$, or alternatively a greedy action is executed with probability $1-\varepsilon$. By decreasing ε over time, the agent progresses towards exploitation.

The Q-learning basic algorithmic iteratively estimates the value of being in a state s and taking an action a, given by $Q(s,a)$. These estimates are computing using

$$\hat{q}_n(s_n, a_n) = r(s_n, a_n) + \gamma \max_{a'} Q_{n-1}\left(s_{n+1}, a'\right), \tag{2.4}$$

$$Q_n(s_n, a_n) = (1-\alpha_{n-1}) Q_{n-1}(s_n, a_n) + \alpha_{n-1} \hat{q}_n(s_n, a_n), \tag{2.5}$$

according to (Powell, 2019), where n is the step, $\hat{q}_n(s_n, a_n)$ is a sampled estimate of the value of being in state $s = s_n$ and taking action $a = a_n$, and where γ is a discount rate. The sampled estimates "bootstrap" the downstream value $Q_{n-1}(s_n, a_n)$. The parameter α_n is a learning rate. The state s_{n+1} is a sampled version of the next state we would visit given that we are in state s_n and take action a_n.

Additionally, RL generally considers a single agent learning in a stationary environment, whereas multi-agent RL considers multiple agents learning through RL. This generalization of the MDP is known as the *stochastic game* (Buşoniu et al., 2010). Multiple RL agents can benefit from shared experience, for instance via communication, but also present some challenges such as the difficulty of specifying a learning goal, the non-stationarity of the learning problem, and the need for coordination (Buşoniu et al., 2010). The need for coordination results from the fact that the effect of any agent's action on the environment depends also on the actions taken by the other agents. Besides, the curse of dimensionality becomes even more severe in multi-agent RL compared to single agent RL (Buşoniu et al., 2010).

In conclusion, reinforcement learning allows agents to adapt to the dynamics of open, complex, and uncertain environments.

2.3. Deep Reinforcement Learning

Deep learning is a subset of machine learning methods based on artificial neural networks (ANNs) with representation learning, in which ANNs adapt and learn from vast amount of data (Bengio et al., 2013). The adjective "deep" comes from the use of multiple layers in what we call the artificial neural network, which are computing systems inspired by biological neural networks that constitute animal brains. We call the ANN a deep neural network when it has multiple layers between the input and output layers.

For large-scale problems with millions of state-actions pairs, storing explicitly all Q-values can become too difficult computationally (Gosavi, 2009). One solution to deal with the curse of dimensionality is Deep reinforcement learning (DRL). DRL extends reinforcement learning by using a deep neural network without explicitly designing the state space, enabling reinforcement learning to scale to problems that were previously intractable (Arulkumaran et al., 2017). The idea is to store all the Q-values in the form of a small number of scalars, where the scalars are the coefficients of regression parameters or the weights of a neural network. Figure 2.2 shows a representation of the interactions between the agent and the environment with policy represented via Deep Neural Networks. Besides, a variant of Q-learning is deep Q-learning, which uses deep convolutional networks.

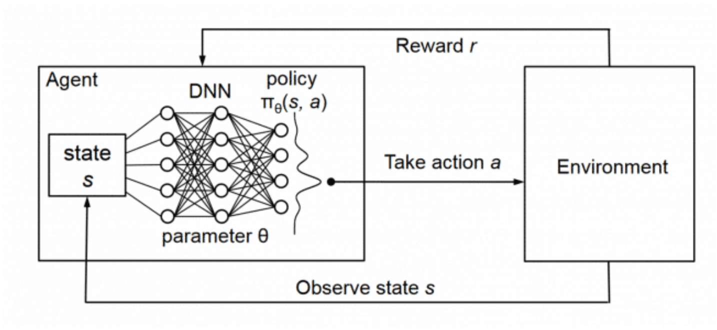

Figure 2.2: Representation of the interaction between agent and environment in Deep Reinforcement Learning. Source: Mao et al. (2016).

2.4. Reward Shaping

One of the main points of concern in reinforcement learning is the definition of reward functions, since the actions that agents will choose are strongly dependent on the reward functions. This is also known as reward shaping or reward engineering, which involves carefully designing reward functions that provide the agents rewards that lead them to a goal (Goyal et al., 2019) and thus to the desired system behaviour (Dewey, 2014). Also, reward shaping has a strong effect on the learning process but it is a difficult and time-consuming task (Matignon et al., 2006; Goyal et al., 2019; Zuo et al., 2020), and there are no strict rules for that (Matignon et al., 2006).

On one hand, as an example, Matignon et al. (2006) explored the influence of some RL parameters such as the reward functions and the initial Q-values over the learning speed. In practice, these parameters are often chosen based on intuition or simply arbitrarily. The authors developed a theoretical study and and also provide experimental justifications for choosing appropriate parameters within the context of goal directed tasks. On the other hand, some research showed that manually shaping reward functions can result in suboptimal system performance and therefore using sparse rewards is often the choice (Chen and Su, 2019). Nevertheless, this is also a relevant matter of concern in reward shaping, since sparse rewards may lead the agents to continue taking random actions for long sequences of actions and hence slowing down learning.

In general, when writing reward functions, we get what we incentivize, not what we intend. Therefore, in many RL problems, the challenge is to make sure whether the reward really reflects what we want the agent to learn. Also, time spent on shaping rewards can make a big difference and save a lot of time in training models.

2.5. Conclusions

In this chapter we have reviewed the theoretical background relevant for this research. We have introduced the Markov Decision Processes, the reinforcement learning technique and how the latter can be combined with deep learning to boost its scalability. Finally, we have introduced the rewards shaping technique and its relevance in reinforcement learning problems.

3 Literature Review

In this section we introduce railway infrastructure modelling and we discuss relevant literature on the Train Routing Problem (TRP) (section 3.1), which is insightful to approach our problem, as well as the TUSP using operations research techniques (section 3.2), followed by applications of deep reinforcement learning applications to operations research problems (section 3.3), including in railway applications and in the TUSP in particular (section 3.4). Finally, we identify the research gap that we approach in this work (section 3.5).

3.1. Railway Infrastructure Modelling and the Train Routing Problem

Zwaneveld et al. (1996) introduced the TRP, which consists of choosing an appropriate route for each train arriving or departing, for given arrival and departure times, in a station area and platform assignment where each train is assigned to one station platform track. While most research on the TRP focuses on train services traversing station areas and not on shunting in particular, the insights we can draw from them are still relevant for the routing part of the TUSP. Nevertheless, we will first have a look at some basic concepts underlying on train routing.

First, a distinction is traditionally made in literature between macroscopic and microscopic models for infrastructure modelling, depending on the level of infrastructure detail. Macroscopic models consider stations as nodes and open track as arcs, whereas microscopic models consider speed limits, curves, gradients, signalling systems, etc. (Radtke, 2014). Therefore, we understand train routing at nodes or station areas as the analysis at the microscopic level, in which the key element to set a route are the track switch or points and the interlocking system. On one hand, a track switch is a mechanical installation enabling railway trains to be guided from one track to another. On the other hand, the interlocking system is an arrangement of points and signals interconnected in a way that changing the elements can only be done in a proper and safe sequence, which prevents setting conflicting train routes (White, 2012). Train routes must be technically protected paths for safe train and shunting movements. We understand conflicting train routes as routes that are not safe from the railway safety principles point of view.

Figure 3.3 shows the microscopic representation of the track layout of Eindhoven. The black continuous lines represent all the tracks used by train services (owned by ProRail), whereas the dashed lines represent secondary tracks that are used mainly for shunting, cleaning and maintenance purposes (owned by NS). The tracks in the shunting yards in [2] and [3] are free tracks since they have two open sides and hence they can be accessed from both sides. Movements over different tracks is possible using the available track switches, which can be identified in the figure as each single track bifurcations. Finally, the grey rectangles in [1] represent platforms from which passengers can board and alight from trains, whereas the rectangles in [2] represent platforms for servicing in the shunting yard.

Second, some authors such as Goverde and Hansen (2013) introduced design parameters for routing such as efficiency, feasibility, stability and robustness. Efficiency relates to minimal running times, whereas feasibility relates to all events being feasible, i.e. conflict-free at the microscopic level. Stability is the ability to absorb initial and primary delays so that delayed trains return to their scheduled train paths without rescheduling.

Finally, robustness is the ability to withstand the stochastic variations of the system without rescheduling rules.

Solinen et al. (2017) elaborated on the indicator Robustness in Critical Points (RCP). The concept of critical points and the related ex-ante indicator RCP can be used to increase timetable robustness. Ex-ante indicators are measures based on timetable characteristics, which can be evaluated at the planning stage without information on the disturbances that may take place. The work by Solinen et al. (2017) introduces a method for RCP optimization with the aim of better understanding the RCP increase at a localised level within a timetable in terms of the effects on the pairs of trains that are part of the indicator. Further work on this methodology could improve the effect on timetable robustness.

Now we have a look at some work that has been carried out on the TRP. The routing problem, decoupled from the timetabling problem, tries to optimize a given objective function in addition to find a non-conflicting route plan. Burggraeve and Vansteenwegen (2017) list possible objectives such as maximizing the amount of trains that can be routed, minimizing the shunting movements, minimizing the average travel time, minimizing energy consumption, robustness, etc. Burggraeve and Vansteenwegen (2017) propose however an integrated approach to routing and timetabling in station areas in which the maximal node usage is minimized and the amount of times a node is used is penalized. The authors define nodes as switches, platforms and network border points.

Bešinović and Goverde (2019) presented the Robust Train Routing Problem (RTRP), as an extension of the TRP by Zwaneveld et al. (1996), which takes into account timetable stability, robustness and capacity assessment. Besides, their model incorporates the idea of using infrastructure resources more evenly, which can have a positive effect on the required maintenance works. The presented model is solved using an heuristic approach.

3.2. Shunting Yards and The Train Unit Shunting Problem

As described in chapter 1, shunting yards consist in complex sets of parallel tracks for sorting and storing trains off the mainline, so that the traffic on the mainline is not obstructed, usually located next to stations where trains start and end their service routes. In literature, two main types of shunting yards based on the track layout are identified (Janssens, 2017). On one hand, the first type is the *carrousel* type of yard, in which most tracks are open on both sides and trains move around the shunting yard along different service locations. An example of this is the Kleine Binckhorst shunting yard, depicted with dashed black lines in Figure 3.1. On the other hand, the second type is the shuffleboard type, in which most tracks are dead-ended, where trains are stacked following the LIFO principle. An example of this is the Cartesiusweg shunting yard, also depicted with dashed black lines in Figure 7.4. Additionally, there is a hybrid layout type without clear characteristics, which is called station layout.

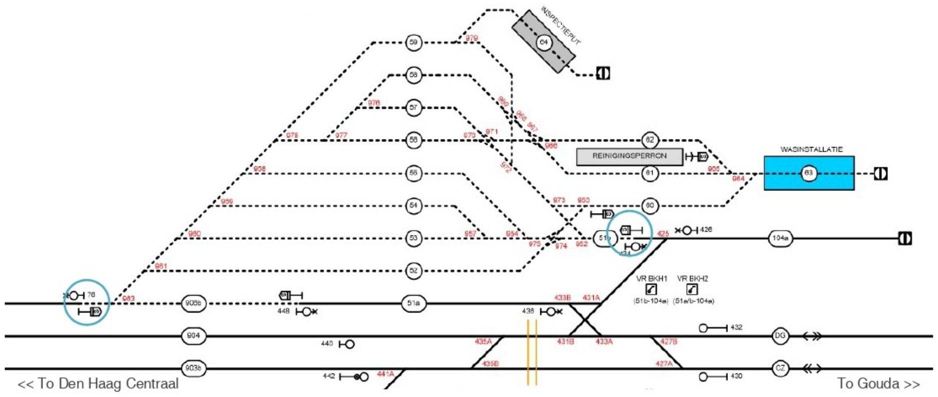

Figure 3.1: Kleine Binckhorst shunting yard (Sporenplan Online, 2019). The entry tracks are marked with blue circles.

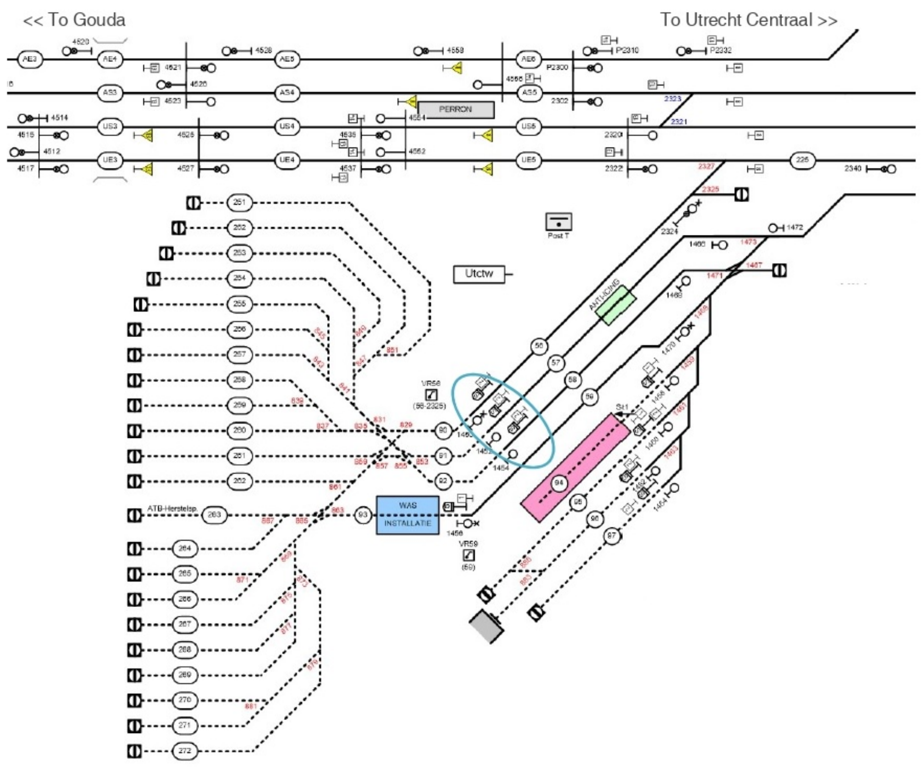

Figure 3.2: Cartesuisweg shunting yard (Sporenplan Online, 2019). The entry tracks are marked with blue circles.

The Train Unit Shunting Problem (TUSP) is the planning problem concerning the assignment of train units from shunting yards to scheduled train services in such a way that the resulting operations are non-conflicting with each other (Haahr et al., 2017). By contrast to the TRP described in section 3.1, the TUSP concerns the activities that are earmarked within the node planning activities, as described in chapter 1, whereas the TRP generally focuses on train services traversing station areas. The main components of the TUSP are matching, servicing, parking, splitting & combining, routing and crew assignment. These components are briefly described as follows:

- **Matching:** train units can be combined in different ways, and it is necessary to find a match between the incoming and outgoing combinations, where each incoming unit is matched to an outgoing unit.
- **Servicing:** a number of tasks have to be carried out on trains, such as cleaning or light maintenance tasks, at specific locations at the shunting yards and before starting a new train service.
- **Parking:** shunting yards at nodes are used to park train units when these are not performing train services so that main tracks are cleared for other running train services.
- **Splitting & combining:** in order to change the combinations of train units, it is necessary to split combinations and/or combine others.
- **Routing:** shunted trains have to be routed from platforms at stations to shunting yards and vice versa, and routed within shunting yards, obeying strict safety rules, so that the matching, servicing, parking, splitting and combining subproblems can be solved.
- **Crew scheduling:** in order to perform the tasks described, it is necessary to assign crew to each task.

It can be observed that routing, in essence, is a rather overarching subproblem since it is associated with the movements that will allow solving the matching, servicing, parking, splitting and combining subproblems, often taking place at various locations within a shunting yard. Therefore, there are clearly strong interactions between the described subproblems of the TUSP and consequently, the individual subproblems of the TUSP are not standalone subproblems and thus they need to be approached in a rather integrated process, which makes it even more complex. The TUSP has been extensively studied in literature, and here we give an overview of some approaches to this problem using classic operations research techniques, particularly Mixed Integer Linear Programming (MILP).

Some authors that researched solution methods for the TUSP showed that several of the TUSP subproblems are already NP-hard (Freling et al., 2005; Lentink, 2006; Haahr et al., 2017), which means that can only be solved in polynomial time.

Defining an objective for the TUSP is not an easy task. According to Haijema et al. (2006), railway planners focus on the following objectives: (1) minimize the amount of shunting work by keeping train units coupled as much as possible, (2) maximize the robustness of a solution by clustering the same type of trains on the same tracks and (3) minimize disturbances that shunting movements might cause to regular traffic. Nevertheless, these goals can be conflicting and therefore planners are satisfied with sub optimal solutions that are feasible.

Furthermore, the scheduling of servicing tasks resembles the flexible Open-Shop Scheduling Problem (OSSP), which is a well-known NP-complete problem. The OSSP consists in processing a set of jobs for given amounts of time at each of a given set of workstations, in an arbitrary order, by contrast to the Job Shop Scheduling Problem in which the order in which the processing steps must follow a specific order. The flexibility feature is a result of multiple machines in which each operation can be processed. In the TUSP, the jobs are the shunted trains and machines are the service facilities. Also, train servicing comes with additional constraints resulting from train parking (i.e. buffer and blocking constraints) and release dates and deadlines resulting from arrival and departure times of trains (Van Den Broek, 2016).

In the context of the department of Research & Development Node Logistics at NS, the physical scope of the TUSP is limited to the shunting yards within nodes (see Figure 1.2), which are owned and managed by NS, as opposed to the infrastructure in station areas, which are owned by the infrastructure manager ProRail. As an example of different nodes areas, Figure 3.3 visualizes the layout of the node of Eindhoven with the station area (1) and two shunting yards (2 and 3). Also, the capacity assessment of the shunting yards is one of the research focus of the department. At the high level, the capacity of shunting yards depends on the train arrival and departure ratios, the arrival and departure time distributions, the incoming train compositions and the assigned service tasks. Therefore, the capacity of a certain shunting yard can only be measured under very specific conditions.

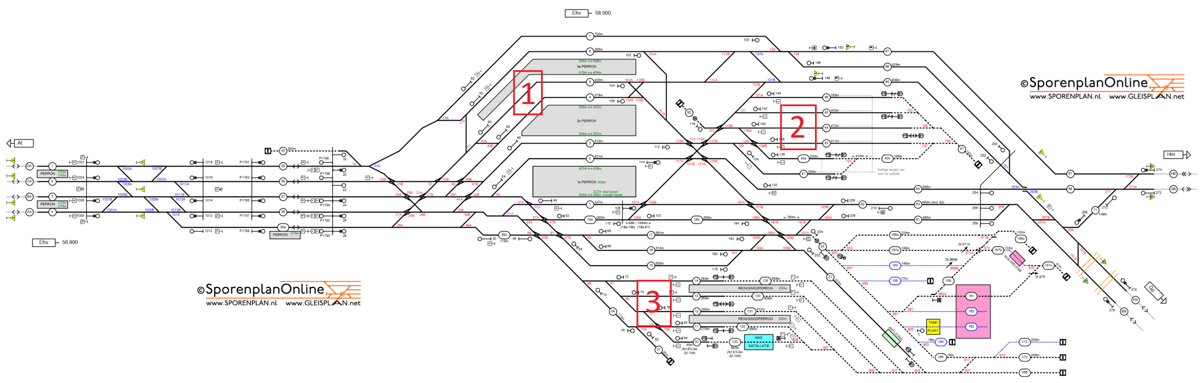

Figure 3.3: Layout of the node of Eindhoven, with the station area (1) and two shunting yards: the Garden (2) and the East-side (3) (Sporenplan Online, 2019).

The definition of the routing subproblem of the TUSP has to be adjusted for the related work carried out in collaboration with NS, which is described in section 3.2 and in section 3.4. Routing is necessary in order to link the parking and servicing activities and it involves selecting a path from a given origin track (e.g. the gateway track) to a selected destination track within the shunting yard. Therefore, two simultaneous decisions have to be made for each shunted train in different steps of the shunting process: 1) select a destination track and 2) find a feasible route from the given origin track to the selected track; so that the resulting route supports the rest of subproblems of the TUSP and a system optimality goal. The work described herein uses the initial definition of routing introduced unless explicitly said the work was carried out in collaboration with NS, which in this case, the work uses the definition of routing as described in this paragraph.

The Train Unit Shunting Problem (TUSP) was first introduced by Freling et al. (2005) (originally published in 2003), which consisted of the matching of arriving trains to departing trains and the parking of these train units on a track while minimizing the number of actions of splitting trains. Follow-up work such as Lentink et al. (2003) extended the model to incorporate train routing to the shunting yard. The authors note that a very large number of possible routes are possible even in normal instances due to the possibility of reversing trains. Besides, in order to find routes for shunted trains, a distinction is made between two classes of algorithms: algorithms that search routes simultaneously and algorithms that search routes sequentially. Simultaneous algorithms are in principles advantageous since they allow computing an overall optimal solution for routing, while taking into account all restrictions and inter dependencies. However, this comes at a high computational cost. By contrast, sequential algorithms have more reasonable computational times and they better resemble the current practice of railway planners, which is convenient from the human-machine interaction point of view.

Kroon et al. (2008) proposed a model to simultaneously solve the matching and parking subproblems of the TUSP, by contrast to Lentink et al. (2003), which approached both subproblems sequentially. Nevertheless, the integration of both subproblems made the amount of constraints to increase dramatically, resulting in high computational costs and hence very limited applicability in real-life cases. In the recent years, NS has developed the *Opstel Plan Generator* (OPG; in English: 'stabling plan generator') (NS, 2018), a tool built to approach the matching, parking, routing and splitting and combining subproblems of the TUSP, which is based on the work by Kroon et al. (2008), with MIP and based on Dijkstra's shortest path algorithm. The OPG determines the matching for arriving and departing train units and the track assignment for parking and the routing based on estimated routing costs, as well as the time and location of train splitting and combining tasks.

While all work on the TUSP reviewed so far used MILP, Van Den Broek (2016), in collaboration with the NS, used a Simulated Annealing (stochastic local search) approach to solve the TUSP, with some aspects being solved by heuristics such as tabu search. It starts with an initial shunting plan, which is improved gradually using a cost function. The results show a high performance of this method and a reasonable computational cost. The results were bench marked against the OPG and the comparison revealed that the approach by Van Den Broek (2016) outperformed the OPG, since it is capable of solving problems with 20 train units reliably and it is also advantageous in terms of computational time. Nevertheless, the local search approach cannot account for the uncertainty of train arrivals and departures and it is not very consistent in terms of results for the exact same instances, which is important for the acceptance as a Planning Assistance Tool by railway planners. Therefore, this approach is quite powerful but it falls short for real-life cases.

An example of an activity graph of a shunt plan is given in Figure 3.4. The nodes represent arrivals (A), movements (M), parking (P and subscript of track number), service tasks (S), departure movements (DM) and departures (D). The solid lines connect nodes concerning the same train, while the grey dashed lines represent the order of movements.

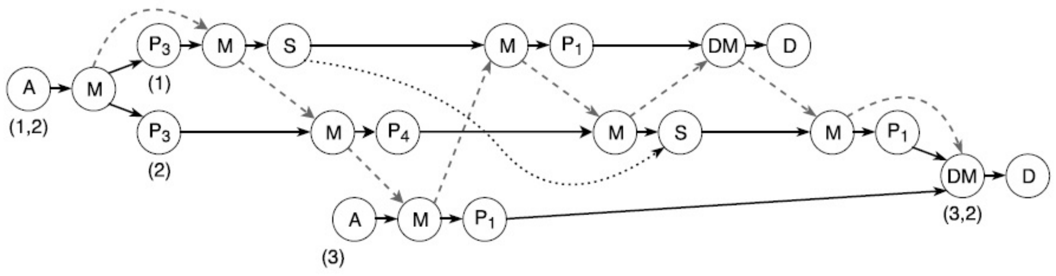

Figure 3.4: Example of an activity graph in the local search framework for the TUSP. Source: Van Den Broek (2016).

Van Den Broek (2016) and Kleine (2019) noted that disturbances in both train arrival times and service tasks often occur in the shunting yard. Therefore, shunting plans are subject to uncertainty, and as a result, the deterministic feasible shunting plans could become infeasible in the operational setting. Consequently, shunting plans should be robust in order to be applicable in real-life cases.

Hence, shunting plans should ideally be able to absorb most disturbances or alternatively require only small adjustments, so that it is not necessary to develop completely new plans. This property of shunting plans is therefore what we understand as robustness in the context of shunting yards. Nonetheless, there is no clear consensus in literature on how robustness can be measured in this context. To improve robustness, Van Den Broek (2016) proposed testing each shunt plan by running a number of stochastic simulations that vary arrival times and tasks duration, and penalize parts of plans that perform poorly in most simulations.

The work by Kleine (2019) proposed a robust, proactive solution method to address uncertainty in the TUSP by means of a Simulated Annealing algorithm. This algorithm takes into account uncertainty during scheduling, such that there is no (or less) need to adapt the shunting plan due to variability in, for example, the arrival time of trains and tasks duration. In the proposed algorithm, a surrogate robustness measure based on the minimum and average slack of the tasks is included in the objective function to guide the search.

Also in the context of robustness in train shunting, Cicerone et al. (2009) researched recovery rules that can be applied to feasible solutions to cope with disturbances in the input data, such as track unavailability or malfunctioning of key resources. The authors have shown the trade-off between robustness and optimality of a solution, as well as the difficulty of creating solutions that are robust to multiple types of disturbances.

Den Ouden (2018), also in collaboration with the NS, extended the work by Van Den Broek (2016) to include the crew scheduling subproblem of the TUSP, using the output of the latter as the input. The author proposed an approach using a greedy heuristic to obtain an initial solution, and a local search improvement step, optimizing a combination of the flexibility, fairness, and walking distances. The method was found to be able to consistently find successful results for realistic scenarios. The work by Van Den Broek (2016) and Den Ouden (2018) significantly contributed towards the Hybrid Integral Planning tool (HIP) that NS is currently developing (Den Ouden, 2019), which aims at integrating the different subproblems and solving them in parallel, by contrast to most previous work which approached the subproblems sequentially. HIP is currently the best performing programme to solve the TUSP.

Haahr et al. (2017) developed a constraint programming formulation, a column generation approach, and a randomized greedy heuristic to approach the TUSP and benchmarked the results against some existing models with real-life instances. The problem approach focused on the parking subproblem, but did not include detailed routing. The randomized greedy heuristic outperformed the other methods developed.

Wolfhagen (2017) proposed an integrated approach for parking, matching and routing with the possibility of relocation of parked train units. We call relocation to those movements from a certain parking track to another parallel parking track via a relocation track where the train reverses its direction, and hence involving a total of two movements. Such movements can let other trains pass, which increases the flexibility of operations in the shunting yard. As a result of adding relocation, the chances of finding a feasible solution increased, but the computational times would increase dramatically for large instances.

Beerthuizen (2018), in collaboration with NS, proposed heuristics inspired in the container stack assignment and relocation problems to improve the occupation rate of shunting yards, in the TUSP context, while maintaining or improving the ability to execute the desired service activities in the yard. The fact that multiple train units on parking tracks can block each other strongly resembles the physical properties of container stacks, in which relocation of containers is necessary to reach those containers that are found underneath. Furthermore, in both problems there is uncertainty in the arrival of containers and trains and, by contrast, the departure times and sequences are known. Nonetheless, there are several differences between both problems. The facilities' utilization rate is higher in rail yards, stacking cranes are needed to move individual containers, while trains are self-propelled compositions of carriages, which cannot be moved individually. Coupling and uncoupling of train units is however possible. Also, there is a wide range of rolling stock types, each with different lengths, while container sizes are much more standardized and container yards often store store a single type of container. Furthermore, tracks in shunting yards usually have various lengths, compared to container stacks which have uniform maximum height, and can often be accessed from both sides, whereas containers in stacks can only be accessed from the top.

In Beerthuizen (2018), two main categories of heuristics are proposed to allocate trains to tracks based on the train characteristics, which are in essence simple decision rules. The first one is the Type-Based Strategy (TBS), which strives to allocate a single type of train on each track, and the second one is the In Residence Time Strategy (IRTS), which allocates trains to tracks based on the departure time to create a chronological departure sequence of trains at each track. A discrete process simulation is used to model the proposed heuristics and applied in two case studies. The results are compared among them and also with previous work at NS (Van Den Broek, 2016; Wolfhagen, 2017; NS, 2018) and show that the proposed heuristics have some limitations such as limited flexibility, not taking into account future events and lower performance compared to more sophisticated mathematical models. Nevertheless, the proposed methods have a high computational efficiency compared to other methods and yield robust results. Besides, the results also provide some insights into factors that influence performance, such as the possible combinations of trains and the shunting yard layout.

Finally, Zhong et al. (2020) presented a MILP model for rolling stock deadheading in an urban rail transit line, which consists in routing before the operation period (i.e. with no passengers on-board), which minimizes the total deadhead mileages, to solve the problem with multiple circulation plans, depots, and rolling stock types. The results showed that deadhead movement of rolling stocks is largely restricted by the depot capacity rather than the switch station capacity, where trains can change their running direction.

3.3. Reinforcement Learning and Deep Reinforcement Learning Applications in Operations Research Problems

Reinforcement learning has been successfully applied in many games such AlphaGo, AlphaZero, Atari games and also in robotics, which increased the attention in other application areas such as transportation, industrial control, education, finance, etc. (Qin et al., 2019). Some specific recent applications of reinforcement learning in transportation include self-driving cars control, traffic lights control, carpooling and ride-sharing platforms.

RL and Deep RL has been applied in the Operations Research domain, but so far not extensively. One example is the research by Mao et al. (2016), who applied DRL to solve a resource management problem. The authors built a system that learn to manage resources directly from experience, after translating the problem of packing tasks with multiple resource demands into a learning problem. The policy is represented as a neural network, which takes as input a collection of images, and outputs a probability distribution over all possible actions. The authors note that the model can support different objectives depending on the rewards setting and that RL-based models are often less explainable, understandable and verifiable than simple heuristics.

Then, we look at the literature on DRL applied to solve optimization problems such as the Vehicle Routing Problem (VRP) and the Travelling Salesman Problem (TSP). The VRP is a combinatorial optimization problem that has been studied in applied mathematics and computer science for decades. The VRP is used to answer the following question: what is the optimal set of routes for a fleet of vehicles to traverse in order to deliver to a given set of customers? It is a generalization of the also well-known TSP, which assumes a single vehicle and unlimited capacity, and both are known to be computationally difficult and are classified as an NP-hard problem. These and other similar problems are studied in the field of combinatorial optimization and solving instances of these problems at the scale of real-world environment is computationally intractable (Toth and Vigo, 2002). Therefore, extensive heuristics have been developed to approach such problems, which produce approximate solutions in polynomial time.

Alipour and Razavi (2015) proposed a method to solve the symmetric (i.e. Euclidian distance, and hence undirected graph) TSP using multi-agent RL, which is used as tour construction heuristic and then the constructed tours of each agent are improved by 2-opt local search heuristic. The experimental results show a high performance of the proposed approach in terms of quality of the solution and computational efficiency.

This contribution is particularly interesting since it incorporates local search to the RL framework to enhance the heuristics of the agent.

Nazari et al. (2018) presented a framework to solve the VRP using RL. The VRP is formulated as a Markov Decision Process (MDP), in which the optimal solution can be viewed as a sequence of decisions. A self-driven learning procedure that only requires the reward calculation based on the generated outputs and following feasibility rules is used. For this, a parametrized stochastic policy is used. Their approach was found to outperform classical heuristics and Google's OR-Tools designed for the VRP and it is able to capture stochastic elements of the problem. The authors note that the proposed framework can easily extended to other VRPs simply by constructing the corresponding state transition function and masking procedure. The masking procedure consists in setting the log-probabilities of infeasible solutions to $-\inf$ or forces a solution if a particular condition is satisfied.

Kool et al. (2018) developed the idea of learning heuristics for combinatorial optimization problems, for which they use models based on attention layers and Pointer Network and train the model using the Reinforce algorithm with a simple baseline based on a deterministic greedy rollout. The experiments focus on routing problems: the TSP, two variants of the VRP, the Orienteering Problem and the (Stochastic) Prize Collecting TSP, which are usually solved by different algorithms. With the proposed method, one has to adjust the input, mask, decoder context and objective function to each problem. The results show that the graph-based method learned strong heuristics for multiple routing problems, getting close to highly optimized and specialized algorithms.

Holler et al. (2019) developed a DRL approach to the Multi-Driver Vehicle Dispatching and Repositioning Problem (MDVDRP). By contrast to the VRP or the TSP, which are static problems, this fleet management and order dispatching problem ride-sharing platforms face has multiple drivers and dynamically changing supply and demand conditions, which requires to account for uncertainty in future supply and demand conditions. The environment rewards the agent for each assigned order with a reward the size of the order price, with reward specifications corresponding to driver-centric and system-centric perspectives. For training, Deep Q-Networks (DQN) and Proximal Policy Optimization (PPO) are used. The learned policies are benchmarked against two kinds of baselines: myopic revenue maximization and myopic pickup distance minimization. The results showed that the driver-centric approach is usually better or at least competitive with the system-centric approach.

RL has been effectively applied in many scheduling and rescheduling problems with proved advantages. With regards to railway operations, applications of RL approaches and DRL in particular are very limited, focusing on scheduling and rescheduling. Rescheduling involves real-time conflict resolution during disrupted railway operations, and consists in reactive traffic management measures on one hand, such as rerouting or reordering trains, and proactive traffic management on the other hand. In proactive traffic management, traffic controllers focus on preventing disturbances rather than solving them, monitoring of current train positions, prediction of train speed profiles or running times, conflict detection, rescheduling of trains in real-time and driver advisory systems (D'Ariano et al., 2014).

One example of scheduling is Khadilkar (2018), who proposed a reinforcement learning algorithm for scheduling railway lines based on a table-based Q-learning. The author notes that the advantages of this approach is its scalability compared to exact approaches, due to decoupling the size of the state-actions space from the size of the problem instance; and the solution quality compared to heuristic approaches, due to the inherent adaptability of RL to solve specific problem instances. In the proposed algorithm, the system consists of a set of block sections connected to form a railway line and the set of actions correspond to binary decisions (stop/go). The Q-values represent the probability of a successful episode termination, which happens when all trains reach their destination. In order to ensure results such that their quality exceeds the output of heuristic approaches, while having comparable online computation requirements, some challenges need to be addressed. For instance, a generalized state space of fixed size is used so that the algorithm can handle different infrastructure and train service instances. Besides, the decisions for individual trains are decentralized and the feature vector is limited to a fixed local horizon around each train so that it can scale to large, realistic railway lines.

In Khadilkar (2018), for the objective function and rewards, the algorithm maintains a threshold of the priority-weighted average delay as the goal to be achieved in each episode. A reward of +1 (success) is given if the sum of the priority-weighted delay is under the current threshold, and -1 (failure) either if it is over the threshold, or if the episode enters deadlock and does not terminate. The reason behind this is to overcome the difficulties of a typical approach consisting in taking delays as the negative reward after completing each move, in order to drive the algorithm towards minimizing the priority-weighted average delay. The integrated reinforcement learning algorithm is driven by a discrete event simulator. The results show that the approach achieved the goal of handling large, realistic problem instances in computation times comparable with heuristic approaches, but with better schedule quality.

Some work has been carried out in the field of railway rescheduling, such as Ning et al. (2019), in which a DRL approach is introduced to approach train timetable rescheduling and minimize the average total delay for all trains along a railway line. The timetable rescheduling considers two possible strategies at the macroscopic level to recover from disturbances: reordering of trains and re-timing. The learning agent is responsible for adjusting running times, dwell times and departure sequences for trains and conflicts are resolved simultaneously. The Deep Q-Network algorithm is used to address the problem. The authors introduce additional constraints to the DRL environment including headways, running times and station capacity. The actual arrival and departure times of trains constitute the agent states, the agent is treated as the train dispatcher who controls the rescheduling process and the action sets are the possible departure sequences for all trains. The reward function is defined based on the negative of the deviation between planned and actual arrivals and departures. Numerical results show that the average total delay outperformed a First-Come-First-Served (FCFS) method.

The Swiss Federal Railways (SBB) explored the application of multi-agent DRL in the Vehicle Rescheduling Problem (VRSP) using "Flatland", a two-dimensional simplified grid environment designed to facilitate experimentation in the VRSP (Nygren et al., 2019; Mohanty et al., 2020), where each agent is assigned to one train. In Flatland, multiple train runs have to be performed while minimizing the global delay on the network. Flatland defines the possible movements between adjacent cells within the grid and the action space, observation space. A visualization of a simple Flatland environment with one train station and one train is given in Figure 3.5. In terms of rewards, agents receive a combined reward consisting of a local and a global reward signal. The reward function includes a negative reward for moving or stopping along the way at every time step until it reaches its target location, a penalty for illegal moves At every time step, agents receive a negative reward. Besides, a global reward is given to every agent if all agents have reached their targets. The reward function includes parameters for tuning collaborative behaviour and creates the objective of finishing the episode as quickly as possible in a collaborative way. Some conventional baselines for the experiments are established, such as stochastic random agents, agents that always choose the “go forward” action or agents that use the shortest path.

Earlier work by Gabel and Riedmiller (2007) noted that although it may be possible to model scheduling problems as a single MDP and thus adopting a global view on the problem, a multi-agent approach can allow performing reactive scheduling including the capability to react to unforeseen events.

Besides, Hirashima (2011) and Hirashima (2012) approached the train marshalling problem using RL, which consist in rearranging freight cars to assemble an outbound train in a freight yard. Both works focus on scheduling methods to rearrange and line cars in the desirable order on the main track using the Q-learning algorithm. Hirashima (2011) focuses on minimizing the total number of freight car movements, whereas Hirashima (2012) focuses on minimizing the total processing time for obtaining the desired arrangement of freight cars for an outbound train.

An overview of the research carried out on RL and DRL in the operations research domain is shown in Table 3.1, which shows each source, the methodology used, the agents system used, the solution approach and the result evaluation.

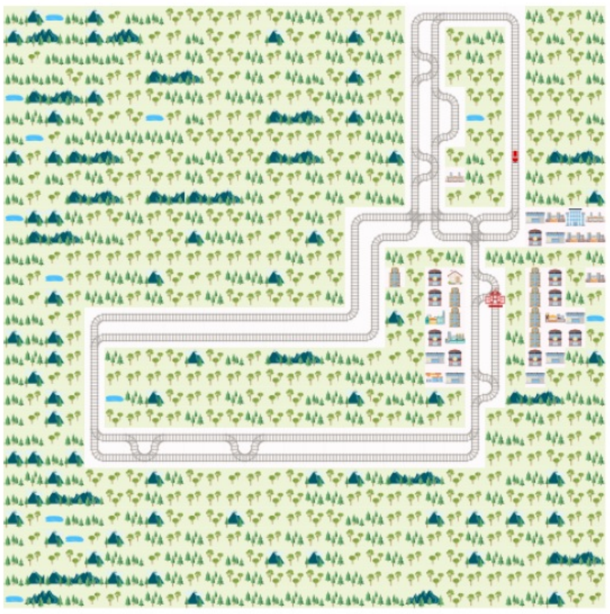

Figure 3.5: Visualization of a simple Flatland environment with one train station and one train (in red). Source: Mohanty et al. (2020).

3.4. Reinforcement Learning and Deep Reinforcement Learning Applications in the TUSP

Although the research on generic routing problems, timetabling and marshalling problems using RL and DRL reviewed in section 3.3 are insightful for the TUSP, their respective approaches have limited applicability on the TUSP since the latter, as described in section 3.2, is a very constrained and context-specific problem. For instance, a relevant similarity is that the TSP and VRP involve visiting a sequence of locations, which is also the case of the TUSP. However, in the TUSP, trains have to stay at each location for certain periods of time, which makes it more complex.

Several authors (Peer et al., 2018; Jamshidi, 2019; Cohen, 2019; Kyziridis, 2019; Barnhoorn, 2020; NS, 2020) have researched the application of DRL to the TUSP in collaboration with NS, with MDPs and with different agent-based models, machine learning algorithms and state representations. The approach by Peer et al. (2018), for instance, is able to account for uncertainty and the results showed it outperformed previous approaches to the TUSP problem, such as the local search approach by Van Den Broek (2016) in terms of consistency of the solutions, but not in terms of solvability of instances. However, one limitation of the work by Peer et al. (2018) is that it is hard to add more functionality to the model using two-dimensional input picture.

Peer et al. (2018) was the very first approach to the TUSP with DRL, in which an event-driven single-agent DRL framework was proposed to approach the matching of incoming train units to outgoing train services, parking and routing subproblems of the TUSP, with (and without) the possibility of train repositioning. The goal was to explore whether DRL could help creating more consistent route plans that could account for uncertainty in train arrival and departure sequences. The framework uses two MDP formulations, an image-like state representation and Deep Q-Network. The state is be represented by a tensor of size $33 \times 33 \times 1$, in which each carriage of a train occupies one pixel. The state contains the following information: 1) number of arriving carriages beyond look ahead, 2) look ahead arrival events, 3) current arriving train, 4) current shunting yard occupation, 5) look ahead departure events and 6) number of departing carriages beyond look ahead. The results show that for both formulations (with and without relocation), the agent is able to find consistent strategies with much lower uncertainty than the local search approach (Van Den Broek, 2016).

Jamshidi (2019) extended the work by Peer et al. (2018) by incorporating service tasks such as cleaning with different state representations. The model is also event-driven but a waiting action (i.e. "do nothing") is added to the actions space so that time constraints associated with service tasks are accounted for. The state

Source	Scope	Method	Multi-agent	Solution approach	Result evaluation
Mao et al. (2016)	Resource management	DRL	No	Neural network, Image-like state	Benchmarked against heuristic approaches
Nazari et al. (2018)	VRP	RL	No	Recurrent Neural Network	Benchmarked against OR-Tools VRP engine
Kool et al. (2018)	TSP, VRP and others	Heuristics	No	Attention layers and pointer network, REINFORCE	Benchmarked against various baselines
Holler et al. 2019	Railway rescheduling	DRL	One agent per driver	DQN, Proximal Policy Optimization	Benchmarked against two myopic baselines
Alipour et al. (2015)	TSP	RL	Each agent constructs a tour	Local search heuristic to improve tours	Benchmarked against five other algorithms
Khadilkar (2018)	Railway scheduling	RL	No	Table-based Q-learning	Benchmarked against heuristic approaches
Ning et al. (2019)	Railway rescheduling	DRL	No	DQN	Benchmarked against FCFS
Nygren et al. 2020	Railway rescheduling	DRL	One agent per train	Ape-X, Proximal Policy Optimization	Benchmarked against three conventional baselines
Hirashima (2011)	Train marshalling	RL	No	Q-learning	Benchmarked against other conventional methods
Hirashima (2012)	Train marshalling	RL	No	Q-learning	Compared performance of their own methods

Table 3.1: Overview of some recent research on RL and DRL in the operations research domain.

resulting from a waiting action applied to the environment is exactly the same as immediately before applying it. Besides, different machine learning algorithms such as Deep Q-network, actor critic and deep value iteration were tested. The latter method was found to perform better than the rest. Nevertheless, the framework was computationally expensive and, as Peer et al. (2018), it does not allow for simultaneous movements, which is actually inevitable in real-life problems and therefore these frameworks cannot be applied directly to real-life cases.

Successive work such as Cohen (2019) moved to a multi-agent system. In this case, the system was composed by two agents, one for the parking tasks and another one for the cleaning tasks. Different approaches were developed, including a basic multi-agent system resembling a single-agent system, and multi-agent systems with full and partial observability. The results revealed a better performance of the single-agent system, by contrast to the initial expectations. Also, the coordination between the parking and cleaning agent was found to be problematic in the learning process.

Kyziridis (2019) also approached the TUSP with service tasks included, although with a single-agent DRL framework. The state is presented as the visual representation by Peer et al. (2018), containing all relevant information of the current time stamp. Policy gradient and Deep Q-network algorithms were tested for agent training. The agent interacts with the environment simulator software built by NS named TORS (*Trein Onderhoud en Rangeer Simulator*, in English: Train Maintenance and Shunting Simulator), suitable for sequential planning. From the results it seemed that Q-learning performed better than policy gradient. Nevertheless, the agent was not capable of learning a stable policy for solving scenarios with increasing number of trains probably due to a too large action space on one hand and the state representation used on the other hand, which worked well in Peer et al. (2018), where service tasks were disregarded.

Barnhoorn (2020) approached the TUSP using an event-driven multi-agent system in which each train unit is treated as an individual agent, addressing matching, including recombination, and parking, with no service tasks. The goal was to analyze whether agents can learn how to combine and park properly. By using multiple agents, the action space significantly decreases to only those feasible actions that a particular train unit can perform. Train departures consist of a list of requested rolling stock types and subtypes without specifying the train unit ID and thus letting the programme choose any train unit ID as long as it matched the train types and subtypes specified for a specific departure, which means that agents had to solve a matching subproblem and learn to cooperate with each other. The model developed gives full information to agents to solve each problem. A set of positive rewards is given to agents depending on the action taken. In general, the perfor-

Source	Method	Algorithms	TUSP subproblems	Logistics information	Result evaluation
Peer et al. (2018)	SA DRL	MDP; DQN	M, P, R	No	Benchmarked against Van Den Broek (2016)
Jamshidi (2019)	SA DRL	MDP; DQN, actor-critic, value iteration	M, S, P, R	No	Compared performance of their own approaches
Cohen (2019)	MA DRL	Markov games; DQN	M, S, P, R	No	Compared performance against SA DRL model
Kyziridis (2019)	SA DRL	MDP; Policy gradient, DQN	M, S, P, R	No	Compared performance of their own approaches
Barnhoorn (2020)	MA DRL	MDP; Value iteration	M, P, R	No	Benchmarked against Van Den Broek (2016)
NS (2020)	MA DRL	MDP; value iteration	M, S, P, R, S&C	No	-
This research	MA DRL	MDP; value iteration	M, S, P, R, S&C	Yes	Benchmarked against HIP and NS (2020)

Table 3.2: Overview of recent research on the TUSP with DRL. M: matching; S: servicing; P: parking; R: routing; S&C: splitting and combining; SA: single agent; MA: multi-agent; DQN: Deep Q-Network.

mance of the model measured as the percentage of correct problem instances solved was significantly lower than that of the local search algorithm by Van Den Broek (2016). It is interesting to note that in such event-driven systems we give agents chances to make the right decisions. Nevertheless, at the same time there is a risk since we are also giving agents chances to make wrong decisions such as undesired movements, which can lead to premature ending of the problem, which is one of the downsides of using reinforcement learning.

All the work reviewed above has contributed towards the development of the current model (NS, 2020), which is mostly based on the work by Barnhoorn (2020), with improvements and extensions with heuristic explorations to stimulate certain actions. Therefore, it features a multi-agent system with one agent per each train unit, MDP formulation and a deep value iteration algorithm. Furthermore, the configuration of the model allows for a high degree of flexibility in terms of problem constraints. It is important to highlight the transition from single-agent models to multi-agent models, from the earliest work on the TUSP with DRL, such as Peer et al. (2018), to the current model by NS (2020). The theoretical advantages of multi-agent systems are the following: (1) they are in general computationally less expensive compared to single-agent models since many computations can be performed in parallel, (2) agents are given smaller action spaces compared to a single-agent (which is in control of "everything") which positively contributes to make the agents converge faster, and (3) they allow for simultaneous movements, which is often the case in real-life instances in shunting yards. An overview of the research carried out on the TUSP with DRL in the recent years is shown in Table 3.2, which shows each source, the methodology and algorithms used, the TUSP subproblems tackled, whether they include logistics information or not and the result evaluation.

3.5. Research Gap and Answer to Sub-Question 1

Extensive research has been carried out to date to approach the TUSP using classic operations research techniques, in some cases being solved as a whole and in some others including only some the TUSP subproblems (e.g. Freling et al., 2005; Lentink et al., 2003; Kroon et al., 2008; Haahr et al., 2017). Earlier work feature sequential approaches to the TUSP subproblems, whereas more recent work have explored approaches integrating all subproblems. Nevertheless, these methods often fall short in real-life problem cases since the mathematical models are too complex or too restrictive in these cases. For instance, there is lack of consistency in the results for similar instances, which makes the output plans hard to be accepted by railway planners. Besides, these models assume perfect information and thus do not account for uncertainty in events such as train arrivals and departures, which further limits its applicability in real-life cases. Aside, Beerthuizen (2018) proposed some heuristics to allocate trains to specific tracks inspired in the container stack and relocation problems to improve shunting movements.

Differently, DRL has been applied to solve train scheduling and rescheduling problems and in the TUSP in particular (Peer et al., 2018; Jamshidi, 2019; Cohen, 2019; Kyziridis, 2019; Barnhoorn, 2020; NS, 2020). Current work in DRL moslty focused in the computational aspects, while detailed logistics of real-life problems and have not been approached so far. Following the promising performances of DRL for solving the TUSP, an overview of the related research carried out is shown in Table 3.2, which shows each source, the methodology and algorithms used, the TUSP subproblems tackled, whether they include logistics information and the result evaluation.

The gaps recognised in the current research are as follows. First, the existing research on DRL do not give guidance to agents to park on specific tracks, to perform movements to solve combination and split problems or to reach certain locations within the yard. Consequently, learning is more difficult and hence takes significantly longer time. Second, they do not account for conflict prevention and resolution, which makes it more difficult for agents to solve instances. Third, they do not prevent unnecessary movements, which we aim at minimizing as much as possible. Fourth, the characteristics of particular shunting yard types are not accounted for and hence it has limited applicability in different yards. Fifth, the efficient use of the parking capacity is not taken into account and therefore it does not maximize capacity. Sixth, there is no distinction between different matching problem types, which again limits its applicability in different problem designs. Lastly, there is lack of insights on how rewards can be shaped to efficiently solve routing problems.

The conclusion of the literature review is used to answer the first sub question: **What information from the logistics side of routing in the Train Unit Shunting Problem can be incorporated into the current multi-agent deep reinforcement learning framework developed by NS?**

We understand this information as routing strategies, in order to minimize the number of train movements and hence to help improving route plans while taking into account the characteristics of each shunting yard type and the type of matching subproblem. This is motivated by the work by Beerthuizen (2018), in which policies from other logistics problems were successfully applied in the TUSP in a different framework. Furthermore, the research on rewards shaping for RL reveal the complexity of designing reward functions such that lead agents to a goal (Dewey, 2014) and there are limited insights on how to reduce the search space for agents by means of these functions (Matignon et al., 2006), which play a significant role in speeding up the learning process of agents (Goyal et al., 2019).

Hence, the scientific gap is the development of routing strategies such that take all the above mentioned considerations into account and to formulate them in the multi-agent DRL framework developed by NS (NS, 2020). At the higher level, the scientific gap concerns the incorporation of information from the logistics side of routing in the TUSP into a multi-agent DRL framework in order to optimize the output route plans and to contribute towards a better learning process by helping agents to make the right decisions from the logistics efficiency point of view.

In relation to the previous, there is also lack of insights on the adaptability and scalability of such routing strategies to fit different problems, such as different shunting yard types or matching problems. This information can be useful for the railway operation point of view since it can explain how generic and flexible can routing strategies be to fit in different train shunting problems. Besides, from the multi-agent DRL framework point of view, it can explain what are the limitations of the framework and what is its applicability to solve other routing problems using DRL.

This is the focus of this research, which constitutes a novel approach to implement routing within the TUSP, in a multi-agent system and deep reinforcement learning. To the best of the author's knowledge, there are no such approaches in related literature. Besides, the applications of artificial intelligence in the field of railway operations are scarce to date.

4 Problem Description

In this chapter we describe the Train Unit Shunting Problem (TUSP) at the conceptual level. We provide descriptions on infrastructure and train modelling in section 4.1 and on each of the TUSP subproblems in section 4.2. Table 4.1 presents the infrastructure notation used for the problem description.

Sets	**Infrastructure**
T	set of tracks
T_g	set of gateway tracks
T_p	set of parking tracks
T_r	set of relocation tracks
T_1	set of LIFO tracks
T_2	set of free tracks
T_s	set of tracks with service facility

Table 4.1: Infrastructure notation used for the TUSP problem.

4.1. Infrastructure and Train Modelling

The physical scope of the TUSP is the shunting yard, which consists of a finite set of tracks $\tau \in T$ connected among them with track switches. Tracks have two sides, which we identify as the A-side and the B-side of each track (see Figure 4.1). We identify the subset of gateway tracks $T_g \subset T$, from where trains arrive and leave from the shunting yard. Besides, we identify the subset of parking tracks $T_p \subset T$, where trains are allowed to park during their stay in the shunting yard, and finally the subset of relocation tracks $T_r \subset T$, which trains can use to move around the shunting yard. The three sets are disjoint and verify $T_g \cup T_p \cup T_r = T$. Parking tracks $T_p \subset T$ can be either free tracks T_2, with both sides A and B open so that trains can arrive from both sides and move towards both directions, or LIFO tracks T_1, with only one side open. Gateway tracks $T_g \subset T$ are always free tracks since they are meant to allow entry and exit, whereas relocation tracks $T_r \subset T$ are usually LIFO tracks. Both sets are disjoint and verify $T_1 \cup T_2 = T$. Each track has a length l_τ and moving over different tracks is possible using the available track switches. Depending on the shunting yard, it may also be possible to identify sets of entry tracks $T_e \subset T$ and exit tracks $T_x \subset T$, which may also be used for temporary parking or sorting of trains. In such case, the following the union of sets should also verify the equality: $T_g \cup T_p \cup T_r \cup T_e \cup T_x = T$.

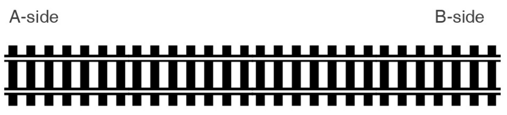

Figure 4.1: Representation of track sides.

Train compositions consist of at least one bidirectional and self-propelled train unit, which is classified according to their type and subtype. The subtype indicates the number of carriages of the train unit, so a train unit type can have more than one associated subtype. For instance, for the Dutch train series VIRM4, 'VIRM' indicates the train unit type, whereas '4' indicates the subtype, which in this case it defines the number of carriages within the train unit. Trains arriving at the shunting yard can be split into individual train units or combined into a train consisting of more train units to match a scheduled train departure. Only train units of the same type can be combined into a single train.

Therefore, we define the set of possible train compositions. For instance, if we consider train units of types VIRM and SLT, with subtypes VIRM4, VIRM6, SLT4 and SLT6, and train compositions consisting of a maximum of two train units, then the set of possible train compositions equals { [SLT4, SLT4], [SLT4, SLT6], SLT4, [SLT6, SLT6], SLT6, [VIRM4, VIRM4], [VIRM4, VIRM6], VIRM4, [VIRM6, VIRM6], VIRM6 } .

Train units may need to have service tasks performed, each of them with a certain duration that depends on the service task to be performed and on the train unit to be serviced. We define the set of scheduled service tasks for a train unit, which belongs to the set of possible service tasks. Service tasks can be either track-specific or track-independent. On one hand, track-specific service tasks require a specific facility located on specific tracks $\tau \in T_s \subset T$ to be performed, such as train exterior cleaning. On the other hand, track-independent service tasks can be performed simultaneously on any parking track $\tau \in T_p$. It can be the case that a service facility serves directly a parking track $\tau \in T_p$, which in this case the set of tracks with that service facility is part of the set of parking tracks, such that $T_s \subset T_p$. However, it can also be the case that the service facility does not serve directly parking tracks $\tau \in T_p$, but dedicated service tracks $\tau \in T_s$, in which case the track sets are independent, as $T_s \not\subset T_p$.

As an example, Figure 4.2 shows the Kleine Binckhorst shunting yard. The shunting yard track layout is depicted with dashed black lines. Switches are shown with red identifiers. The subset of gateway tracks $T_g \subset T$ consists of two tracks (marked with blue circles in Figure 4.2.The subset of parking tracks $T_p \subset T$ consists of tracks 52 to 59, which are all free tracks and thus belong to the set T_2. Additionally, tracks 61 to 64 are service tracks and thus belong to the set T_s, where tracks 61 to 62 are also free tracks belonging to the set T_2 and tracks 63 and 64 are LIFO tracks and thus belong to the set T_1. Track 63 also forms the set of relocation tracks T_r. In this case there is an intersection between the sets of LIFO tracks T_1 and the set of service tracks T_s, but it should be noted that a LIFO track can simply be a relocation track with no service facility, and the other way around, a facility can also be placed on a free track $\tau \in T_2$. The union of tracks 52 to 64 form the complete set of tracks T.

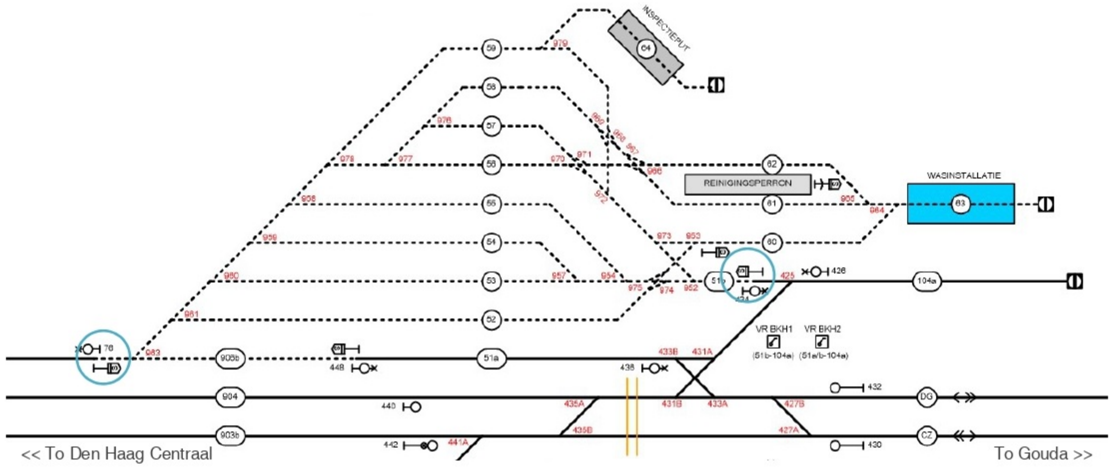

Figure 4.2: Kleine Binckhorst shunting yard (Sporenplan Online, 2019). The entry tracks are marked with blue circles.

4.2. Train Unit Shunting Problem Subproblems

As introduced earlier, the subproblems of the TUSP include matching, routing, parking, servicing, splitting and combining and crew scheduling. In this section we cover each of these subproblems, except of crew scheduling, which for simplicity it is not taken into account in this research. Firstly, we present a summary of the different subproblems of the TUSP described with their constraints and objectives in Table 4.2.

TUSP Subproblems	Main constraints	Objectives
Matching	(1) Total number of arriving train units and departing train units is equal (2) Departure time of each train unit not before having completed all scheduled service tasks after arrival	Match arriving train units with departing train units
Routing	(1) Routes are always non-conflicting (2) Each claimed route is blocked during the duration of the train movement using it	Minimize the number of train movements and maximize robustness of solutions
Parking	(1) Parked train lengths do not exceed the physical available track length (2) Each train unit parked on track τ has an unobstructed route to leave when it has to depart	Park each arriving train unit on parking tracks $\tau \in T_p$
Servicing	(1) Track-specific tasks can only be carried out at specific tracks (2) Each service tasks has a specific (3) One service task can be carried out at most on a train unit at a time	All service tasks for each train unit must be completed during stay at shunting yard
Splitting and Combing	(1) Train units can only be coupled when parked next to each other on the same track τ (2) Only train units of the same type can be coupled together	Minimize number of coupling and splitting operations
Overall TUSP	Arriving and departure tracks and time for each train must be respected	

Table 4.2: Overview of the TUSP subproblems.

4.2.1. Matching

Arriving and departing trains are composed of one or more train units of the same rolling stock type. The goal of the matching subproblem is to find a mapping between each arriving and departing individual train unit at the shunting yard, which will constitute an injective and surjective set of train matchings. First, we assume that the total number of arriving train units of each type and subtype equals the number of departing train units of the same type and subtype so that train unit flow conservation is ensured. Second, the scheduled departure times for all trains take place later than all arrival times, with sufficient time in between to perform movements and to complete the scheduled service tasks, so that all matchings are feasible in time. Also, in this formulation we assume that the shunting yard is empty at the beginning of each problem instance and so it does at the end of each.

Figure 4.3 shows an example with a list of 5 arriving trains and a list of 3 departing trains. The lists are not ordered in time but they must verify the conditions described above, so that all matchings are feasible in time. The plus sign represents a composition of train units in one single train. The arrows represent all feasible matchings between arriving train units and departing train units. Note that each matching involves a train unit of the exact same rolling stock type and subtype. The objective of the matching subproblem is to select one of the feasible matchings for each train unit, so that the mapping is injective and surjective. One possible solution is represented in Figure 4.4.

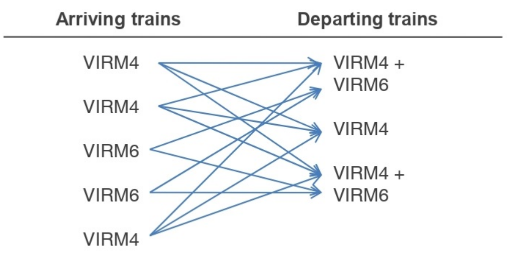

Figure 4.3: Representation of feasible matchings for a particular instance.

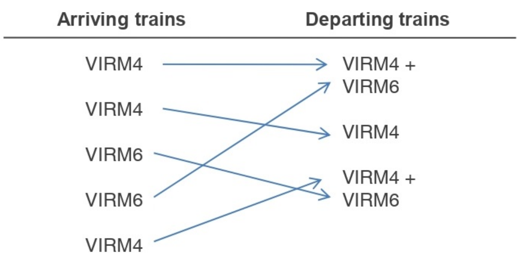

Figure 4.4: Representation of one possible solution to a matching problem.

4.2.2. Routing

The routing subproblem of the TUSP involves finding feasible route plans for all trains. We define a route as an ordered sequence of track sections and switch sections (we refer later to these as track parts). More specifically, routing in the shunting context involves simultaneously (1) selecting a destination track within the shunting yard from the set T from a given origin track (which is the gateway track for the first movement of each arriving train at the yard) and (2) finding a feasible route from the origin track to the selected destination track. The system objectives are (1) minimizing the amount of shunting work and (2) maximizing the robustness of the solutions. For the first objective, we define shunting work as the amount of train movements. We understand a train movement as an action in which a train changes its position in the shunting yard, from one origin track to one destination track, without changing its direction on the way, i.e. without performing any saw movement.

Simultaneous train movements in time in the shunting yard are allowed as long as the routes used are non-conflicting, which means each track part can only be used at most by one train. During the execution of a train movement, all track parts within the route are blocked (see the business rules in Appendix C), which means that they cannot be used by any other train until the movement finishes. When the movement finishes, the track parts traversed are released and can be used by other trains. Therefore, one of the main objectives of finding non-conflicting routes is to find routes with the minimum number of switches in order to minimize conflicts between requested routes and hence to maximize the robustness of the route plan. Additionally, minimal usage of switches may also have a positive effect on infrastructure maintenance costs.

Furthermore, in general trains can only move from track parts that allow for parking or saw movements to track parts with the same characteristics. On the other hand, the duration of a route is based on estimations based on the number of tracks and switches traversed and the number of necessary saw movements. We call a saw movement the manoeuvre of reversing the direction of a train, in which (1) the train moves to a track in which saw movements are allowed, then (2) the driver walks to the opposite end of the train, where it takes over the train controls again and (3) drives the train away in the front-facing direction (which is opposite to the original direction) (see Figure 4.5). Saw movements may be necessary to access certain tracks within the shunting yard. However, since this type of manoeuvre takes some time, railway planners try to minimize them as much as possible.

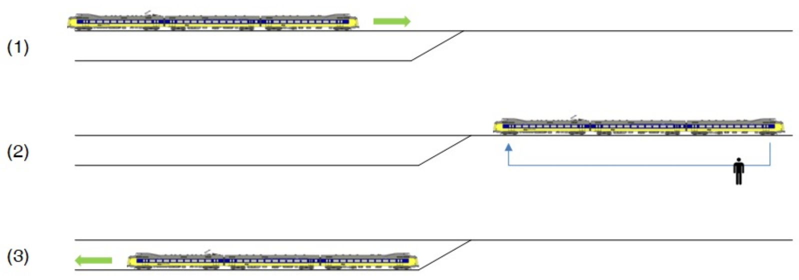

Figure 4.5: Representation of a saw movement.

4.2.3. Parking

The main objective of the parking subproblem is to park the trains arriving at the shunting yard in front of one of the two sides of the sequence of trains already parked in the track τ. There is a strong interaction between the parking subproblem and the routing subproblem since in order to prevent trains blocking each other and thus hampering routing, a parking strategy is needed.

The main constraints of this subproblem are that (1) the train length does not exceed the physical available track length of a parking track τ and that (2) each train parked on the track τ has an unobstructed route to leave the parking position when it needs to depart according to the scheduled departure sequence. Nevertheless, it should be pointed out that it may be allowed to temporarily park a train on a different track to let another train leave or arrive, so that the flexibility to prevent conflicts is increased. Furthermore, the possibilities to park trains on a track $\tau \in T_p$ are constrained by the type of track τ, since in LIFO tracks $\tau \in T_1$ one side is inaccessible and hence there is only one side possible for parking. However, in free tracks $\tau \in T_2$ both sides may be accessible and the parking side will depend on the arrival side of the train. Splitting and combining trains is in general only allowed on parking tracks.

4.2.4. Service Tasks

Shunting yards are used not only to park trains, but also to carry out some service tasks on them such as cleaning, inspections and minor maintenance. For each train unit we are given a set of service tasks that need to be carried out during the stay at the shunting yard (or alternatively, during the instance), which we assume that they can be executed in any order, i.e. with no precedence relations between tasks. The execution of each service task has a specific duration, which depends on the train type and subtype to be serviced, needs specific resources and in some cases can only be carried out at dedicated facilities. Therefore, we make a distinction between track-specific service tasks and track-independent service tasks. Therefore, since in the context of the TUSP there are limited resources to carry out servicing, all tasks need to be scheduled in such a way that all trains can depart on time with all service tasks completed.

In the case of the Nederlandse Spoorwegen (NS), there are 35 service sites around the network where service tasks can be carried out (Den Ouden, 2018). NS calls these service tasks as *first-line service*, which include cleaning and performing inspections on the train units, and take place on a daily basis, generally overnight but also during off-peak hours. The regular types of activities are the following:

- Safety check A: Set of inspections performed every twelve days. An A-check takes 8 to 27 minutes depending on the type of the EMU. These checks can be performed on any parking track.
- Safety check B: Set of inspections, performed every two days, depending on the type of the EMU. A B-check takes about 38 to 90 minutes to be completed, depending on the type of EMU. These checks can be performed on any parking track.
- Internal cleaning: Cleaning of the interior of the train by a cleaning team. This happens every day and takes between 24 and 46 minutes per carriage, depending on the type of the EMU. These need to be performed at a track with a cleaning platform.
- External cleaning: Cleaning of the exterior of the EMU using a washing installation, either with soap or with oxalic. This is done once a week. Every ninth time is with oxalic, the rest is with soap. Oxalic cleaning takes 4 minutes per carriage, and 10 minutes for each end. Cleaning with soap takes 1 minute per carriage, and 10 minutes for each end. External cleaning is done in a dedicated external cleaning facility.

Besides these, there are some activities that are carried out on an as-needed basis.

- Small repairs.
- Removing graffiti.
- Service requests.
- *Automatische Treinbeïnvloeding* (ATB) maintenance.

4.2.5. Splitting and Combining

In TUSP real-life problem instances it is often the case that some of the trains in the set of arriving trains do not match trains in the set of departing trains and therefore in these situations it is necessary to split and/or to combine train units in order to solve the matching. It is important to note that trains must always consist of train units of the same type, but coupling train units of a different subtype is possible. The direct consequence of this constraint is that we can only couple trains of the same type. In general, in real-life cases, trains do not consist of more than 2 or 3 train units, depending on the rolling stock type of which they consist. Furthermore, it should be noted that in order to couple train units, the target train units must be parked next to each other on an appropriate track, facing train unit fronts with each other. Since this condition constraints more the overall TUSP, it is desirable to minimize train couplings as much as possible and hence to limit them to the strictly necessary ones only.

5 Methodology

In this chapter we describe the methodology that we follow in this work, based on a multi-agent system in a deep reinforcement learning framework modelled as a Markov Decision Process (MDP), which is a type of probabilistic sequential decision model, with value iteration. We present an overview of the TORS-MATDRL framework (section 5.1), the agents (section 5.2), the environment with the state representation and rewards (section 5.3) and the neural network and the policy evaluation algorithm (section 5.4). This work builds on the framework developed by Barnhoorn (2020) and NS (2020) and therefore the methodology description in this section is mostly retrieved from these works.

5.1. Overview

Figure 5.1 shows a high level representation of the TORS-MATDRL framework for the TUSP. The framework inputs include (1) the set of problem instances, (2) the location and (3) the business rules, or constraints. Firstly, the problem instance is defined by a list of arriving and departure trains with their scheduled arrival and departure time, its composition of train units and the matching and service tasks assigned for each train unit. Secondly, the location is defined by the model of the shunting yard. Thirdly, the business rules consist of the set of constraints imposed by NS for train shunting. The complete list of business rules can be found in Appendix C. The inputs described are feed into the core programme, which is described below.

NS developed two programmes to model the TUSP as an MDP, which interact with each other: on one hand, the simulation environment TORS (*Trein Onderhoud en Rangeer Simulator*, in English: Train Maintenance and Shunting Simulator), which is the simulation environment for sequential planning. On the other hand, the Multi-Agent Train Unit Deep Reinforcement Learning (MATDRL), which is a programme that uses DRL to solve planning problems for shunting yards. It does so by treating each train unit as a separate agent, and lets the agents choose the actions for their train unit. The agents choose which action to take based on deep value iteration. We use a neural network to tackle the curse of dimensionality, as described in detail in section 2.3. TORS is mainly responsible for generating the location and actions. Both programmes are coded in Python.

Each attempt to solve an instance is called an episode. The main loop of an episode is described as follows:

- An instance is randomly selected from the batch of input instances.
- For each significant event a trigger is generated.
- The instance loops over these triggers; at each trigger, the agents get a chance to perform an action.
  - Arrival: new agents for the arrived train units are generated and passed to the Network and Map used by all agents.
  - Otherwise: The agents choose actions using value iteration and perform them.
- When all agents have performed an action or when it is the starting time of the next trigger, the next trigger starts.

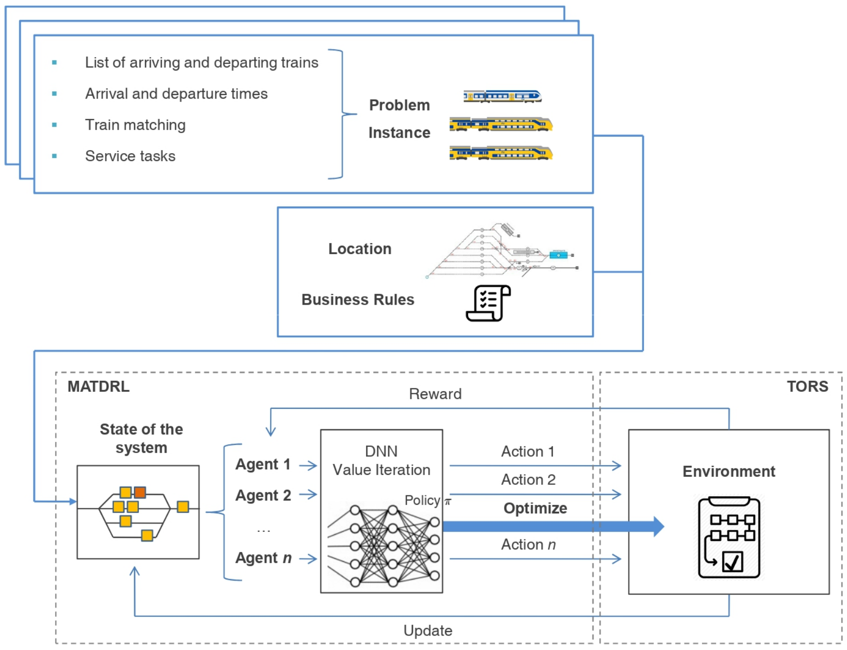

Figure 5.1: High-level representation of the multi-agent DRL framework for the TUSP.

- At the end of the instance the results are logged and a new instance begins.

This loop is broken when a violation takes place and a new episode starts.

5.2. Agents

Each train unit is associated to an agent, which will make decisions on behalf of this train unit. The objective of the TUSP in the multi-agent DRL framework is to maximize the number of correct train shunting departures, which is the agents' goal. By correct departures we mean that each train (1) departs on time from the right track in the shunting yard, (2) with the correct composition (with split and combination if necessary) and (3) with all scheduled service tasks completed, subject to constraints (i.e. *business rules*) such as the physical layout of the node, non-conflicting movements, the location of service task facilities and arrival and departure times of trains.

Each train unit has the following attributes associated:

- Union: the identifier of the train of which the train unit is a part.
- ID: the (unique) identifier of the train unit.
- Type: the train unit's rolling stock type.
- Subtype: the train unit's rolling stock subtype.
- Connections: the IDs of the train units that are in the same train.
- Position: the position of the train unit in its train.
- Service times: how much time the train unit needs for each service task.
- Directions: the directions in which the train unit can move.

- Match index: index assigned according to the position of the train unit in the departure sequence.

In previous work on the TUSP with DRL at NS such as Barnhoorn (2020), train departures only consisted of a list of requested rolling stock types and subtypes without specifying the train unit ID and thus letting the model choose any train unit ID as long as it matched the train types and subtypes specified for a specific departure. However, it is also possible to solve the problem by specifying the train unit IDs for all train departures. At NS, in practice, for planning of daily operations a detailed list of departing train units is given. By contrast, for the assessment of the capacity of shunting yards train departures only consist of a list of requested rolling stock types and subtypes. This makes a significant difference in whether the matching subproblem has to be solved or not.

The approach with free train unit IDs (0% matching) is less constrained than the approach with fully known departing train unit IDs (100% matching) and hence more flexible but it has negative effects on learning since agents have to learn to do solve the matching problem. By contrast, the approach with fully known departing train unit IDs is more constrained and as a result it speeds up learning as agents are already told what matching they have to do and hence agents do not have to learn to solve this subproblem.

With 100% matching, the match index works as follows: the train units forming the exact requested composition of the (current) next departing train are assigned a match index 1, the train units within the next train requested to depart are assigned a match index 2 and so on. It is important to note that the match indexes are not assigned to individual train units but to train compositions. Therefore, if two specific train units form the requested composition for a departing train, then both train units will be assigned the same match index according to the position of the departing train in the departure sequence. Besides, if a train composition does not match any requested train composition for a departing train, in which case it may need to split and then to combine with other train units, then all train units of which the (current) train consists of will be assigned a match index equal to zero.

With 0% matching, the match index works in a very similar way but with the difference that all train units that match the requested rolling stock types and subtypes for a particular train departure are given the same match index, also according to the position of the train in the departure sequence. As a result, with this approach there are much stronger dynamic effects in terms of matching since every time a train departs, the match indexes of the train units remaining at the shunting yard are updated in a way that certain trains can suddenly become blocked by another agent.

The agents' action space is discrete, $A = \{1, ..., k\}$, and has the following possible actions, which are also represented in Figure 5.2:

- Arrive: lets a train arrive at the location, reserving all track parts on its route.
- Move: move a train from one track to another specific track.
- Set back: reverses the driving direction of a train.
- Combine: combine two train units of the same rolling stock type into one single train, either on side A or side B.
- Split: split a train into two train units, either on side A or side B.
- Service: perform a service action on a train unit.
- Wait: let a train wait till the following action occurs. It does not change the current state.
- Depart: let a train leave from the shunting yard into the passenger service.

The action triggers are the events that prompt agents to act. The following triggers are used:

- Arrival: generated for each arriving train.
- Departure: generated for each time at which a train should depart.

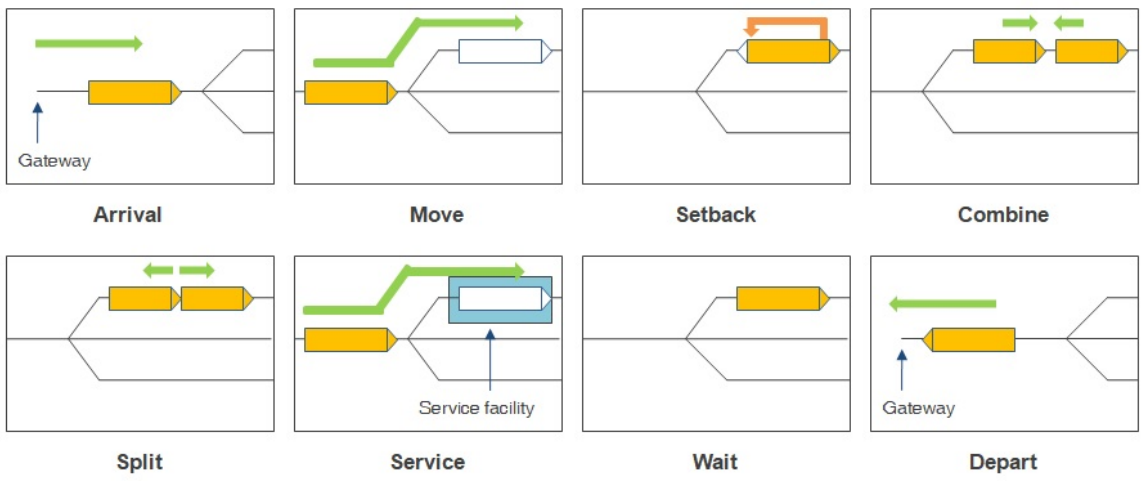

Figure 5.2: Actions representation.

- End Service: Generated at the end of a service task to allow trains to move towards and away from the service facility.
- Middle: generated in between *Arrival*, *EndService* and *Departure* triggers, depending on the setting of the TORS-MATDRL framework, to allow for combination, splitting, service tasks and relocation.

At any given action trigger, only a subset of actions is available. For instance, not all tracks may be reachable in a given state and thus only a subset of *move* actions will be available. Also, combination of trains is only possible when there is a train unit of the same type (or correct ID, depending on the context) next to the agent's train and on the same track, which must be a track where parking is allowed. Additionally, servicing can only be started if the train unit has at least one service task assigned and it is parked on a track where this service task is allowed to take place. The overall idea is to mask out all invalid actions so that agents can focus on relevant actions only.

Therefore, in this methodology, the distribution and importance of actions in the action space is highly imbalanced. In many situations, only a very few actions are available, while in other cases there are many options and the decisions are highly critical to the task's success. These characteristics of our methodology results in a more complex exploration during training.

Agents can still end commit certain violations after which the current episode will end prematurely. The most common violations are the following:

- Train cannot leave: at the time of departure, the corresponding train was not on the gateway track.
- Did not depart: at the time of departure, the corresponding train was on the gateway track but the agent did not choose to depart, or another train was blocking the exit side of the departure track.
- Composition not present: at the time of departure, the corresponding train composition had not been arranged yet.
- Arrival track reserved: entry track length was exceeded upon train arrival due to presence of trains parked on the entry track.

An example of the initial part of a decision tree with triggers and actions going on in time in a shunting yard for one train unit is given in Figure 5.3. It is easily noticeable that the decision tree grows exponentially in each action trigger for a single train unit, which illustrates the phenomena of the curse of dimensionality, as mentioned in chapter 2.

The agent accesses the neural network in order to compute its state values using the Value Iteration algorithm. Agents follow an ε-greedy policy to perform actions when they are prompted to do so. As explained in chapter 2, the training of the network starts with $\varepsilon = 1$, which means that agents select possible actions randomly. During the training process ε decays gradually, which means that agents will become more likely to select a greedy action.

When agents are prompted to perform a greedy action, they evaluate the values of all the states and the received rewards resulting from all the possible actions. The values are computed by a forward pass through the neural network and the rewards are given by the environment. The agent with the highest sum of the value and the reward will be allowed to perform its preferred action first, which will be the one that yields the highest value. The goal of the agent is always to receive the highest possible reward at the end of the episode. After an action is performed, the experience is stored in the memory of the network. The experience is a tuple containing the starting state, the resulting state and the received reward.

5.3. Environment

The environment of an agent comprises non-agent objects, which the agent can observe and act upon. It handles the state representation and the reward function and it contains all information about the problem in the framework. Since the environment models an MDP, agents only take into account the current state of the system, while previous states are disregarded.

The state of the shunting yard is given to the agents in the form of a one-dimensional array. In this form, it avoids sparsity and it is scalable in terms of problem extensions. The array is split into parts based on the type of information handled, each with fixed size so that the input is always consistent in terms of size and placement of information. In particular, the array is split in the following parts:

- **Shunting yard:** State of the shunting yard, with all information about the train units placed on their position on a track and track reservations.
- **Arrivals:** Future arrivals with their rolling stock, length, and time until arrival.
- **Own train:** All information about the train unit of the agent; the composition of its train, the track on which it is parked, any rolling stock types it needs to connect to to match a future departure, whether it already matches a future departure and current and next trigger.
- **Departure request:** Information about the current departure request; what rolling stocks are requested and whether the agent matches the requirements.
- **Future departures:** Information about future departure requests; the rolling stock types needed, the time and where they should depart.

The pseudocode in Algorithm 1 shows how the assignment of rewards is made in detail, and the values of the rewards are presented in Table 5.1. The way the rewards are designed allow for a gradual learning of the sequence of actions in one instance, which is relevant for tasks such as servicing and splitting and combining. Each agent receives local and global reward signals. In particular, agents individually receive rewards for correct combination or splitting (4)(6), rewards associated with starting and completing a service task (8)(10), for a movement towards departure track before a departure trigger (13) and for departure with service tasks completed (16) or departure with service tasks not completed (18). The last train to depart correctly completes the instance and receives a reward that accounts for this system-wide accomplishment (2). Finally, in any other case, a reward equal to zero is given to the agent (20).

It can be observed that these rewards give agents limited guidance for routing within the shunting yard, which mostly target the events described above but do not target the prevention of trains blocking each other and an efficient use of the infrastructure capacity in the yard.

Additionally, an example of a simple instance with 3 train units in Heerlen (see its track layout in Figure 5.4) is shown in Table 5.2. There are no service tasks in this instance and train units 9401 and 8601 need to combine

Algorithm 1 Heuristics for Assignment of Rewards

```
Input: list of agents, train unit, action chosen, action trigger.
Output: feasibility reward.
1: if the action completed the instance successfully then
2:     return Reward.All_left
3: else if action is Connect and train will become any correct composition then
4:     return Reward.Comb_correct
5: else if action is Split and train will become any correct composition then
6:     return Reward.Split_correct
7: else if action is Service and assigned service tasks are not completed then
8:     return Reward.Start_clean
9: else if event is EndService and train unit corresponds to the event and action is Move or SetBack and
   current time corresponds to the event then
10:    return Reward.End_clean
11: else if event is PrepDeparture and action is Move and train is next to depart then
12:    if destination track is gateway track then
13:        return Reward.Move_depart
14:    end if
15: else if action is Depart and service tasks are completed then
16:    return Reward.Leave_clean
17: else if action is Depart then
18:    return Reward.Leave_dirty
19: end if
20: return 0
```

Action	Description	Name	Reward
Service	Start service task	Reward.Start_clean	0.5
Service	Complete service task	Reward.End_clean	0.5
Combine	Combine into correct configuration	Reward.Comb_correct	0.5
Split	Split into correct configuration	Reward.Split_correct	0.5
Move	Move towards departure track before departure trigger	Reward.Move_depart	0.5
Depart	Depart without completing all service tasks	Reward.Leave_dirty	0.1
Depart	Depart with all service tasks completed	Reward.Leave_clean	1.5
Depart	All trains have departed	Reward.All_left	8.0

Table 5.1: List of rewards and their values in TORS-MATDRL.

to depart correctly. A feasible solution for the instance in TORS-MATDRL is shown in Figure 5.5. The figure visualizes the time dimension in the horizontal axis, the action triggers, the actions taken by each agent and the rewards received. It is interesting to highlight that all *Middle* triggers are located around the *Arrival* and *Departure* triggers, as explained in section 5.2. Trains arrive at the shunting yard on track 84 and move towards the parking tracks 26, 39, 40 or 44, but they can also use track 0. It can be observed in the instance that train unit 9401 moves around the shunting yard unnecessarily before connecting with train unit 8601 (from 40 to 0 and then back to 40). Therefore, both Algorithm 1 and Figure 5.5 illustrate some of the limitations of this framework. A more comprehensive list of limitations is presented as follows:

- The framework does not prevent conflicts between train units.
- The framework does not account for conflict resolution, when conflicts have happened.
- In general, the framework does not give preference for parking on specific tracks.
- The framework does not make any distinction between different types of tracks, i.e. gateway tracks, parking tracks and relocation tracks.
- Movements around the shunting yard prior to departure are not guided.

- The framework does not prevent unnecessary movements.

From a computational point of view, as a result of the limited guidance to agents, it is difficult for agents to learn a well performing policy.

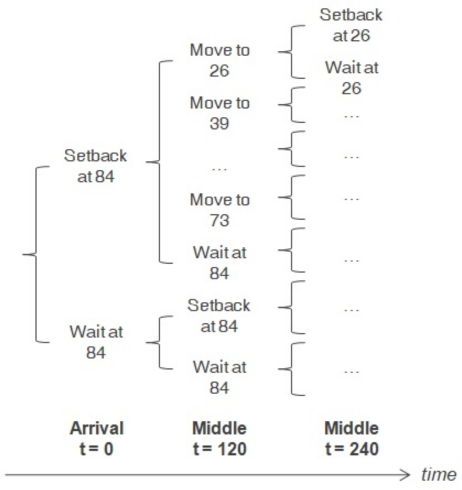

Figure 5.3: Example of the initial part of a decision tree with triggers and actions going on in time in a shunting yard for one train unit. Train units arrive at track 84. We assume that all setback and moving actions take 120 seconds and therefore the next trigger takes place 120 seconds after such action can be taken.

	Event	Train units ID	Time	Track
1	Arrival	[9401]	48,551	gateway
2	Arrival	[8601]	59,074	gateway
3	Arrival	[2401]	62,470	gateway
4	Departure	[8601, 9401]	76,752	gateway
5	Departure	[2401]	81,451	gateway

Table 5.2: List of arrival and departure events for an example instance. Time is given in seconds.

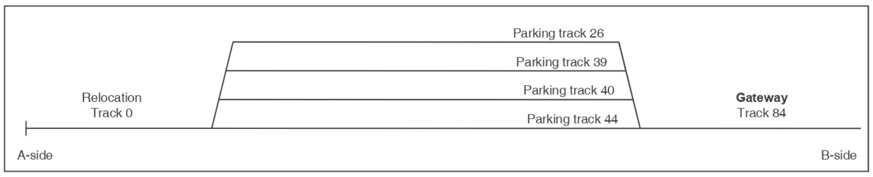

Figure 5.4: Simplified representation of the shunting yard in Heerlen.

5.4. Neural Network and Algorithm for Policy Evaluation

All agents share the same neural network, which is advantageous from the computational point of view since the amount of parameters significantly decreases and the experiences of each agent can be stored in the replay memory, which means the network can learn from the diverse experiences of all agents, which still receive personalized inputs. For more details refer to Barnhoorn (2020). The input layer of the neural network is a single array whose size depends on the number of arriving train units in an episode as described in section 5.3 and the network is build using a sequential model with 5 hidden layers, with 512, 256, 128, 64 and 32 and nodes. All nodes use the Rectified Linear Unit (ReLU) activation functions. Additional information on

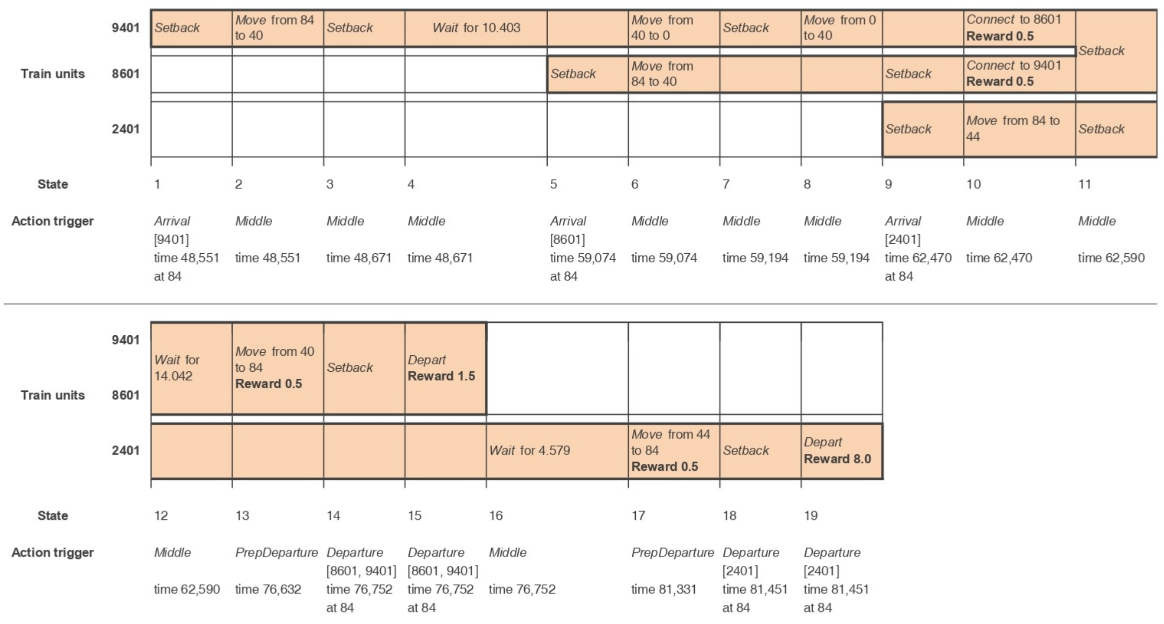

Figure 5.5: Example of an instance solved with 3 train units in Heerlen. Notes: track 84 is the gateway track and setback time is set to zero.

ReLU can be found in Glorot et al. (2011). The output layer is also fully connected and has one node representing the value of the state. Furthermore, there is a target network with the exact same architecture whose weights are synchronized with the main network with a certain rate of episodes in order to keep the values stable.

Following the descriptions by Barnhoorn (2020), the network is prompted to train on a batch of experiences from its memory, which is called as experience replay. This is done by sampling experiences from its memory, with a preference for more recent experiences. The target network calculates a value for the resulting states, and the sum of these values and the corresponding rewards are used as targets for the main network. The main network calculates values for the starting states, and then computes the loss with respect to the targets. Finally, it performs a gradient descent step on the weights using this loss. The loss function that is used is Adaptive Moment Estimation optimizer (Adam). Adam is an algorithm for first-order gradient-based optimization of stochastic functions, based on moving averages estimates of lower-order moments. Additional details on Adam can be found in Kingma and Ba (2015).

Value Iteration creates a policy by computing the value of the policy $V_{n+1}(s)$ at state s by exploring all actions that can be taken from that state onwards and iteratively improving the estimate $V_n(s)$. The algorithm initializes $V_0(s)$ and repeatedly updates the Q-value $Q_n(s,a)$ and the value $V_n(s)$ until a predetermined convergence condition is met (Equation (5.1) and Equation (5.2) according to Powell (2020)). Please refer to Barnhoorn (2020) for the detailed algorithm of value iteration in DRL to select an action.

$$Q_{n+1}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_n(s') \qquad \forall s \in S \tag{5.1}$$

$$V_{n+1}(s) = \max_{a \in A} Q_{n+1}(s,a) \qquad \forall s \in S \tag{5.2}$$

6 Problem Approach

In this chapter we present the extensions that we develop in this work for the TORS-MATDRL framework as our routing optimization solution according to the scientific gap identified in chapter 3. Firstly, we develop an Heuristic for Random Exploration (section 6.1) to efficiently search the state-action space and hence to speed up the learning process. Secondly, we develop Heuristics for Routing Strategies for the train units in the shunting yard and we formulate them by means of reward functions (section 6.2). The routing strategies aim at preventing and tackling conflicts between trains, helping agents to cooperate to solve combination problems, minimizing the number of movements, making a more efficient use of the infrastructure capacity in the shunting yard and hence to improve the performance and quality of route plans. The routing strategies also feeds the heuristic for random exploration to further enhance them. The routing strategies developed include a type-based routing strategy (TBS) and an in-residence time-based routing strategy (IRTS). Figure 6.1 shows the extensions implemented on top of TORS-MATDRL.

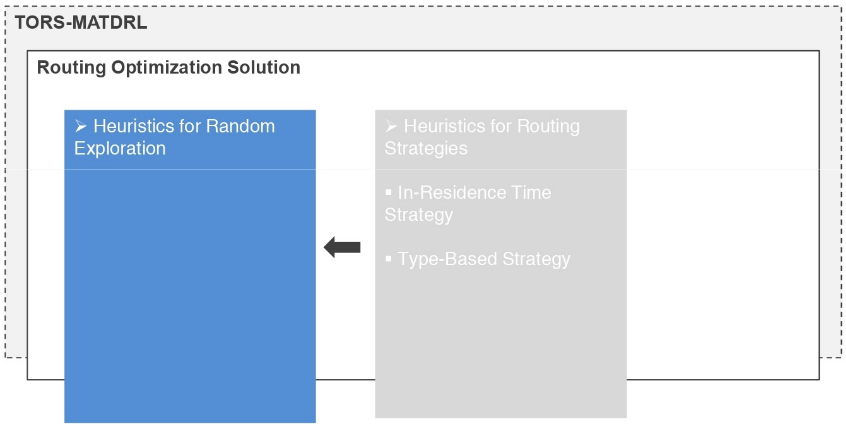
TORS-MATDRL
Routing Optimization Solution
➢ Heuristics for Random Exploration
➢ Heuristics for Routing Strategies
▪ In-Residence Time Strategy
▪ Type-Based Strategy

Figure 6.1: Extensions implemented on top of TORS-MATDRL.

In general, the development of implementations have been carried out starting with very simple instances, i.e. with a very limited number of trains, no service tasks and no need for combination or splitting. These experiments are used to make adjustments in the heuristic for random exploration, initial routing strategies based on the original rewards (Algorithm 1) and tuning the learning parameters. After this, we gradually increase the complexity of our experiments by adding larger sets of instances, more agents and including service tasks, splitting and combination. After obtaining new trained models, we test them and assess their performance. Based on this we make adjustments in the implementations for the same set of instances (high frequency loop) and repeat the process until we are enough satisfied with the results. In this case, we upgrade the problem to a higher complexity (low frequency loop), for which we generate a new set of instances and repeat the process of continuous improvement. The conceptual model of the process described is shown in

Figure 6.2. The experiments that we carried out are therefore used to answer our research questions and sub questions, and are presented in chapter 8.

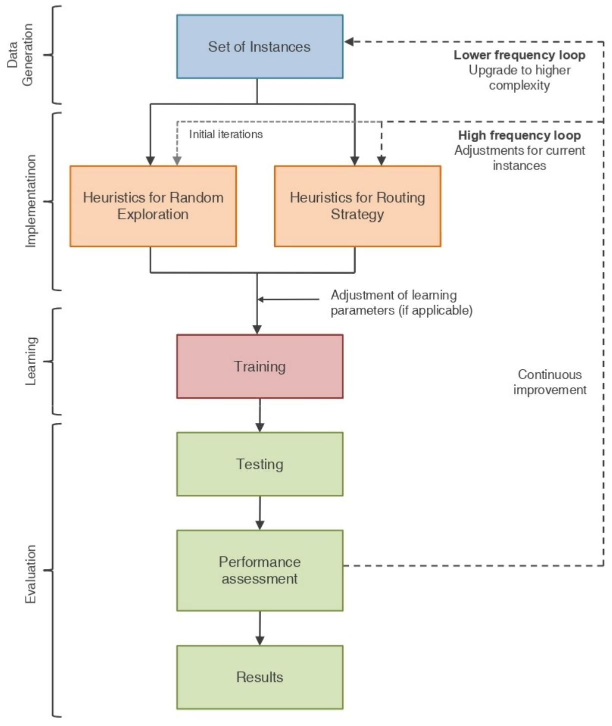

Figure 6.2: Conceptual model of the development of implementations on top of TORS-MATDRL.

6.1. Heuristic for Random Exploration

The neural network training starts with random exploration, which means that agents select an action from the set of possible actions randomly, at each time trigger. An exploration phase is necessary to find random initial points in the search space. The extent to which we allow agents to explore randomly depends on the number of instances and their complexity (e.g. number of trains) and it should be large enough so that it explores the search space sufficiently. Should random exploration be too small, it will immediately get into local minima. Also, the duration of random exploration is based on the number of steps, which represents each time an action is chosen by an agent, rather than the number of instances, since the number of steps represent how much memory is filled. The number of episodes could be misleading since larger instances fill up the memory faster.

Ideally, we would like to explore the entire search space by means of purely random exploration. Nevertheless, the technical requirements for this are very high in terms of computational costs, computational power and memory. Actually, even for small instances with around 10 action triggers, the chances that agents are able to solve the complete instance with purely random exploration are very low and the solvability, understood as the percentage of instances correctly solved out of the total amount of instances processed, usually remains at 0% after trying to solve hundreds of instances. This becomes impractical in our research due to the computational costs and time limitations. Additionally, as noted in section 5.2, the imbalance of actions in the action space result in more difficult exploration.

Therefore, in order to make it more practical for this research, we incorporate an exploration heuristic to stimulate certain actions, based on the principles of large neighbourhood search by Shaw (1998). This heuristic will allow agents to efficiently explore the state-action space, as well as to identify and reason about key decisions along their way.

Such heuristic basically gives a higher weight to some actions in certain situations to be chosen by agents, on top of a baseline weight. This weight is the sum of (1) the reward given by the random exploration heuristic and (2) the actual reward, which agents learns in reinforcement learning. Therefore, it is important to note this interconnection between the heuristic for random exploration and the reward heuristics. In other words, the routing strategies presented below in section 6.2 are part of the heuristic for random exploration. Finally, it is important to note that agents do not learn the rewards given by the random exploration heuristic; these are just meant to guide the agent during random exploration to explore the search space more efficiently.

Note that the assignment of actual rewards, which is an input to the heuristic for random exploration, is made according to the corresponding function for rewards assignment. For instance, this could be the original rewards as presented in Algorithm 1. New functions to assign rewards associated with each of our routing strategies are presented further below in section 6.2. Therefore, in relation to this, it should be noted that the heuristic for random exploration is built as a generic extension of the routing strategies that do not contain strategy-specific elements. Additionally, the heuristic for random exploration are generally independent from specific types of shunting yard and hence they are meant to apply for all scenarios.

Preliminary to the criteria, we introduce the concept of unnecessary movements, which we understand as movements of agents that do not have any goal such as clearing the gateway track, parking, performing a service task, clearing a service facility, connecting with another train or moving to the gateway track for departure. In other words, we can describe an unnecessary movement as a movement that will yield no immediate reward. In relation to this, we introduce the following list of criteria:

- We prefer an agent choosing the action *Set back* over *Waiting*.
- We prefer an agent choosing the action *Wait* over an unnecessary movement.
- We strongly prefer the next train to depart to move towards the gateway track in the *PrepDeparture* trigger.
- We strongly prefer the next train to depart to either *Wait* or *SetBack* when it is already on the gateway track in the *PrepDeparture* trigger.
- We want to stimulate the agents to perform a *Split* or *Combine* action when they do not match any requested train composition for a departing train.
- We want to stimulate agents to perform their assigned service tasks.
- We want to stimulate agents to depart when they are ready to do so.

We developed the heuristic presented in Algorithm 2, based on the general criteria above. For these, we assume there is one single gateway track $\tau \in T_g \subset T$. The heuristic presented below has been developed gradually as the result of extensive testing on instances with increasing complexity. In general, the heuristic for random exploration significantly helps increasing the performance during random exploration in terms of the frequency of trains combining correctly, starting service tasks and departing correctly. For simple instances it can even be possible to solve some instances using this heuristic. The heuristic is not able to solve instances with higher complexity due to the increasing number of action triggers and hence lower probability to solve them.Nevertheless, it is important to note that this heuristic focuses more on helping agents identifying certain patterns such as the ones mentioned and listed above in the criteria list.

Lastly, this methodology tends to constraint the search space into a local area, and therefore it becomes an ill-posed problem that might fall into local minima. This is its main disadvantage, by contrast to pure random exploration, which leads to global optima. The main advantage of the heuristic for random exploration is however that it significantly helps speeding up the learning process.

Algorithm 2 Heuristic for Random Exploration in TORS-MATDRL Pseudocode

```
Input: list of agents and current state
Output: chosen action.
1: Initialize list of rewards
2: for All possible actions do
3:     Compute actual reward
4:     reward ← actual reward + 0.05
5:     if action is Move and train is the next to depart then
6:         if there are still trains to arrive at gateway track then
7:             if destination track is gateway track $\tau \in T_g$ and departure is scheduled before next arrival then
8:                 reward ← reward + 0.5
9:             else if destination track is gateway track $\tau \in T_g$ and departure is scheduled later than next arrival then
10:                reward ← reward - 0.5
11:            end if
12:        end if
13:    else if action is Wait and train is the next to depart then
14:        if agent is at gateway track $\tau \in T_g$ and departure is scheduled before next arrival then
15:            reward ← reward + 0.25
16:        else if agent is not on gateway track $\tau \notin T_g$ and end time of action exceeds time of scheduled departure then
17:            reward ← reward - 0.5
18:        end if
19:    else if action is Wait and train is not the next to depart then
20:        if agent is at gateway track $\tau \in T_g$ and end time of action exceeds time of next scheduled departure then
21:            reward ← reward - 0.5
22:        end if
23:    else if action is Wait and there are still trains to arrive at gateway track $\tau \in T_g$ then
24:        if agent is at gateway track $\tau \in T_g$ and end time of action exceeds time of next scheduled arrival then
25:            reward ← reward - 0.5
26:        end if
27:    else if trigger is PrepDeparture and train is the next to depart and action is Move then
28:        if destination track is gateway track $\tau \in T_g$ then
29:            reward ← reward + 0.75
30:        end if
31:    else if next trigger is Departure and train is the next to depart and action is Wait and agent can move in both directions then
32:        if agent is at gateway track $\tau \in T_g$ then
33:            reward ← reward + 0.75
34:        end if
35:    else if trigger is Departure and train is the next to depart and action is Setback then
36:        if agent is at gateway track $\tau \in T_g$ then
37:            reward ← reward + 0.75
38:        end if
39:    else if action is Depart and train is the next to depart then
40:        reward ← reward + 0.5
41:    else if action is Split or Connect and train is not in the correct composition for a departure then
42:        reward ← reward + 0.5
43:    else if action is Setback then
44:        reward ← reward + 0.5
45:    else if action is Wait then
46:        reward ← reward + 0.05
47:    else if action is Service then
48:        reward ← reward + 0.5
49:    end if
50:    if reward <= 0 then
51:        reward = 0.01
52:    end if
53: end for
54: Choose weighted random action (based on rewards).
```

6.2. Routing Strategies

In this section we introduce our routing strategies, which aim at overcoming the limitations of the work by NS (2020) as described and shown in section 3.5 and section 5.3. In particular, as described above, our routing strategies aim at preventing and tackling conflicts between train units in the shunting yard, helping agents to cooperate to solve combination problems, minimizing the number of movements, making a more efficient use of the infrastructure capacity in the shunting yard and hence to improve the performance and quality of route plans. Our routing strategies are based on the parking strategies introduced by Beerthuizen (2018), which are the Type-Based Strategy (TBS) and the In-Residence Time Strategy (IRTS). Train parking and sorting is only one part of the full problem and hence these are extended in this work to create complete routing strategies consisting of four components: (1) standard parking rules, (2) combination and split rules, (3) conflict resolution rules and (4) unnecessary movements rules. We describe them for a generic shunting yard layout and will be adapted in chapter 7 for specific types of shunting yards. We formulate the routing strategies in our TORS-MATDRL framework by means of reward functions so that the learning of agents of such strategies is stimulated.

Routing in the TUSP involves, in general, one train arriving at the shunting yard and then moving around service facilities (if it has service tasks scheduled there) and parking tracks for storage, and finally depart from the shunting yard. Therefore, in the TORS-MATDRL framework, routing takes place around the *Arrival*, *End-Service* and *Departure* event triggers. It should be noted that event triggers are system-wide and hence all agents may be prompted to perform an action, although some of them such as the *EndService* event trigger are associated with the event of a specific train unit completing a service task. *Middle* triggers, by contrast, are not associated with any specific train unit.

From the routing perspective of this research, we are particularly interested in analyzing the interactions of routing with those service tasks that have limited resources such as cleaning tasks, which can only be carried out at dedicated facilities, and hence require additional movements between tracks within the shunting yard. Agents need to learn how to handle such resource-limited tasks properly so that the facilities can be used by all trains requiring it and allowing all trains to depart on time. By contrast, those service tasks that can be carried out at any parking track do not really have an impact on the resulting shunting plans since they do not require additional movements. Therefore, in normal circumstances, such service tasks should not affect the shunting yard infrastructure capacity. For this reason, we do not consider them.

Besides, we make the following assumptions for generic shunting yards: there is a set of parking tracks $T_p \subset T$ in the shunting yard, there is one single gateway track $\tau \in T_g \subset T$ in one side of the parking tracks (B-side), from where all trains arrive and leave from the shunting yard, and a set of relocation tracks $T_r \subset T$ on the opposite side of the parking tracks (A-side).

It should be noted however that, as described in section 3.2, the shuffleboard type of shunting yards have their parking tracks $\tau_p \subset T$ open on one single side and hence for such shunting yards the assumption of having relocation tracks $T_r \subset T$ on the opposite side of the gateway track $\tau \in T_g$ is not applicable. In order to take this difference into account in the conceptual description of the routing strategies presented in this section, we indicate what adjustments are needed to fit each routing strategy in a shuffleboard type of shunting yard.

Additionally, each list of trains parked on a parking track are read from A-side to B-side. We refer to such lists as parking queues, where the train parked on the edge of the queue on the A-side is at the back of the queue, and the train parked on the edge of the queue on the B-side is at the front of the queue (Figure 6.3). Also, Figure 6.3 serves as a representation of what we understand as a generic shunting yard, with n parallel parking tracks.

6.2.1. Preliminar Observations on Rewards Shaping

It is interesting to note that earlier work on the TUSP in collaboration with NS (Peer et al., 2018; Jamshidi, 2019; Cohen, 2019; Kyziridis, 2019) worked with negative rewards for movements, relocations or faulty movements, but Barnhoorn (2020) noted that this strategy had a negative impact on the learning process due to negative values of the states. The adjustment of the reward functions in such a way that rewards were always equal or greater than zero and thus ensuring positive state values had a positive effect in the learning process.

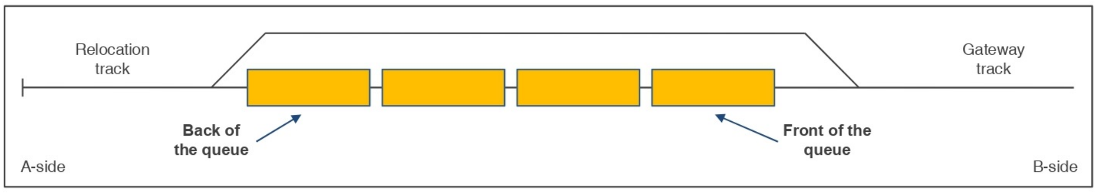

Figure 6.3: Representation of the conventions used for routing strategies.

Nevertheless, small negative rewards can still be helpful to discourage agents to take undesired actions without making the state values too negative, so we will take advantage of this technique.

For the implementation of routing rewards, its better to give rewards as much as possible since there are many different actions and state changes in one single instance, which makes the reward very sparse, and these are also very often delayed. In general, agents learn better when they can be assigned a reward immediately after they perform an action. Nevertheless, as mentioned in section 2.4, the challenge is to make sure whether the reward really reflects what we want the agent to learn.

Besides, we keep the essence of the rewards introduced in section 5.3, although we will adjust the values of some of them, and specific types of shunting yards may require further adjustments in the conditions under which such rewards are assigned. The underlying idea is that the original rewards target the main events of an instance and that we do not want to distort the behaviour of agents towards an undesirable way. In relation to this, it is important to understand that the different values of the rewards give an indication of the priority of each state resulting from a specific action. In other words, there is a hierarchy of actions in terms of what is our priority for agents to do during an instance, and hence to learn during training. Therefore, the rewards that we incorporate in the framework have to be designed in line with these observations.

For instance, we want to give priority to splitting, correct combination and servicing events over a correct parking event under the assumption that such a sequence of events associated with one agent during an instance will be easier to be learnt. Therefore, we want a train with service tasks assigned to start servicing as soon as it arrives at the shunting yard, so that it clears the service facility as soon as possible. With respect to splitting, we want trains that have to split to do so as soon as possible after arriving at the shunting yard (or right when entering the parking area, depending on the shunting yard type) so that each individual train unit can move to an appropriate parking track. With respect to combination, we want trains that need to be combined to do so as soon as all train units involved in the matching are present in the shunting yard. Among different moving actions, we also want to prioritize those that will allow to perform a correct combination or to start a service task over those that will not allow this to happen at the destination track. It should be noted that only some of the agents in each instance will have to perform both of these actions and hence it is safe to assume that both have the same priority and we can give agents the freedom to decide which action to perform first, which will highly depend on the context. Also, in some states it could be the case that only servicing can be performed, for instance when the train unit to be combined with has not arrived at the shunting yard yet.

Additionally, since our framework includes the possibility to relocate trains to correct parking queues, similar considerations have to be made. In general, assuming that agents learn how to properly park according to our routing strategies, we can also assume that relocation will only be necessary as a corrective measure. The design in the reward function will prompt relocation under certain conditions. Nevertheless, it is necessary to make sure that the total reward obtained by the agents involved in a relocation movement is not higher than the reward obtained in a situation in which the same agents parked correctly in the first place. In this way, we will prevent agents from incorrectly learning a routing strategy in which incorrect parking plus relocation would be preferred over correct parking in the long term.

Lastly, the reward for a correct parking should not be higher than the reward given for a correct departure with all service tasks completed. This is because the ultimate goal of the agents is to depart and finish the episode, which should be rewarded higher than successful parking. A higher reward for successful parking would make the agents focus in performing actions such that lead to the desired parking strategy rather than focusing on departing correctly on time. In consequence, the agents behaviour would deteriorate and slow down learning. As a result of the previous considerations, we update the reward values as shown in Table 6.1.

Action	Description	Name	Reward
Service	Start service task	Reward.Start_clean	1.0
Service	Complete service task	Reward.End_clean	0.5
Combine	Combine into correct configuration	Reward.Comb_correct	1.0
Split	Split into correct configuration	Reward.Split_correct	1.0
Move	Correct parking	Reward.Park	0.5
Move	Move towards departure track before departure trigger	Reward.Move_depart	0.75
Depart	Depart without completing all service tasks	Reward.Leave_dirty	0.1
Depart	Depart with all service tasks completed	Reward.Leave_clean	1.5
Depart	All trains have departed	Reward.All_left	8.0

Table 6.1: List of rewards and their values.

6.2.2. Type-Based Strategy

The Type-Based Strategy, as described by Beerthuizen (2018) is based on the categorization of arriving train compositions on train unit type in such a way that trains of the same train unit type are stacked on the same tracks as much as possible. The goal is therefore to have all trains grouped by train type in the shunting yard. The underlying assumption behind this parking strategy is that it can help minimizing the number of movements in the shunting yard when there is need to combine train units, and hence to solve the combination problem. Figure 6.4 gives an example of parking according to this strategy. It should be noted that, as explained in section 4.2, combination of train units is only allowed between train units of the same type, although it is allowed with different subtypes.

It is important to note that the Type-Based Strategy, as described, only makes practical sense when there is no predefined matching towards departing train units (0% matching), which means that each departing train consists of a list of train types and subtypes required, but no specific train unit IDs. Under these conditions, agents have to solve the matching problem, and the Type-Based Strategy can help them to find the correct agent to combine with if applicable. By contrast, under the conditions of predefined matching (100% matching), there is no matching problem and hence not really useful to stack trains of the same type on the same tracks ignoring the train unit IDs, as agents will look for the exact train unit ID to combine with instead, if applicable.

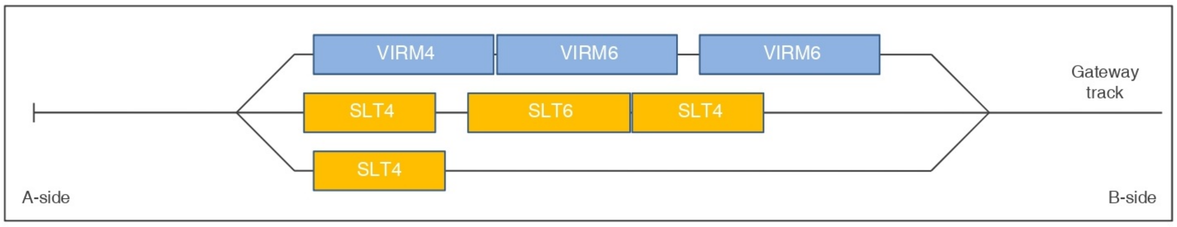

Figure 6.4: Example of the TBS standard parking rules with trains of types VIRM and SLT.

In the TORS-MATDRL framework, the basic idea is to reward agents for parking on a track $\tau \in T_p$ directly next to a train unit of the same type k. Therefore, with the TBS standard parking rules, one would expect to see trains of a single type on each parking track $\tau \in T_p$. Note that this rule does not take into account the train unit IDs. We say that the TBS parking rules are violated when there are trains of different types parked on the same parking track $\tau \in T_p$. It is also important to note that this routing strategy does not guarantee that trains can depart a priori without conflicts. The main reason is because as trains are arriving and being parked, we cannot anticipate which exact train unit IDs will depart on each train in the departure sequence and hence there are relevant uncertainties associated not only to train arrivals but also to departures. As a result, we potentially need to include conflict resolution strategies.

The TBS-based Routing Strategy consists of the following four components:

1) Standard parking rules: In this routing strategy, in *Middle* and *EndService* event triggers, we reward those movements that result in the train arriving from the B-side to a parking track $\tau \in T_p$ parking directly next to a train unit of the same type k (i.e. the train unit parked in the front of the queue) (1). When this is violated, a negative reward is assigned instead (2). It should be noted that on dedicated facilities for servicing, service tasks have priority over the standard parking rules. Therefore, train movements associated with servicing in dedicated service facilities are carried out regardless of the resulting queue of parked trains on the destination track, if applicable. By contrast, for those service tasks that can be carried out on any parking track, the standard parking rules still prevail.

2) Combination and split rules: Due to the relaxation of the matching problem, the amount of possibilities for combination and splitting among train units can potentially increase significantly. Nevertheless, as explained, there are also uncertainties associated to train unit departures. Therefore, in general, we cannot anticipate whether a specific combination or splitting operation is actually convenient to solve the instance or not. As a result, the solution proposed will only reward positively or negatively those that are most likely to be convenient. We say that a unit is not in the correct composition when the composition does not match any future departure request.

On one hand, with regards to combination, positive rewards will only be given only for those units that are either (1) originally in a composition that does not match any future departure or (2) such that after combining with another certain unit it will become a composition matching the requirements for the next departure. For any other combination, a negative reward will be given. Also, combination is allowed on any track $\tau \in T$. On the other hand, when it comes to splitting, it will only be rewarded positively to a train unit if its original composition does not match any future departure. By contrast, a negative reward is given either (1) if the original composition of the train unit was already matching the requirements for the next departure or (2) if at least one of the compositions resulting from the splitting operation does not match the requirements for any future departure. For any other splitting scenario, a negative reward is given. Splitting is only allowed on parking tracks $\tau \in T_p$. This design of the combination and splitting rewards with positive for correct operations and negative rewards for any other scenario is necessary to prevent agents from overexploiting the correct actions just to collect reward.

3) Conflict resolution rules: During the attempt to solve an instance, certain situations can happen in which the strategies already presented are insufficient to drive agents to do the correct sequence of actions to solve the instance. Such situations are mostly the result of the uncertainties associated to train departures. Under the TBS standard parking rules, there are chances that we can find the right composition for the next departure parked in the front of a queue of trains, but it is also often not the case. For such cases, we introduce the following strategies for conflict resolution: (a) late combination on gateway track and (b) relocation. Relocation consists in moving a train from one parking track $\tau \in T_p$ to another parking track $\tau \in T_p$ using a relocation track $\tau \in T_r$.

The key preconditions to trigger these strategies for conflict resolution are the following: (1) there are no trains in the front of any queue (gateway side) on any parking track $\tau \in T_p$ in the shunting yard such that are in the correct composition for the next departure, (2) it is neither possible to solve this incompatibility for the next departure by simply combining or splitting train units parked in the front of one of the queues, and (3) there are no arrivals left. The latter is intended in order to manage the uncertainties associated with train departures, which is also linked to the fact that conflict resolution only applies for the next departure. Finally, it is interesting to note that agents need global vision on the shunting yard in order to solve the conflicts described.

Figure 6.5 illustrates two conflicting scenarios. In (1), the next departure is a composition consisting of one single SLT4 train unit, but there is no such composition in the front of the queues on the parking tracks $\tau \in T_p$.

Nevertheless, it is possible to obtain such composition by simply splitting the composition SLT6+SLT4, which are coupled on track 2. It would also be possible to relocate the SLT4 train unit parked in the back of track 2 (A-side), but splitting is preferred since it will require much less movements to depart. On the other hand, in (2), the next departure consists of two train units, VIRM4+VIRM6. Such composition is present on track 1 but not in the front of any of the queues, which means that it is blocked on the gateway side (B-side). In this case, since there are no other candidate train units, the only possible solution is to relocate the composition VIRM4+VIRM6.

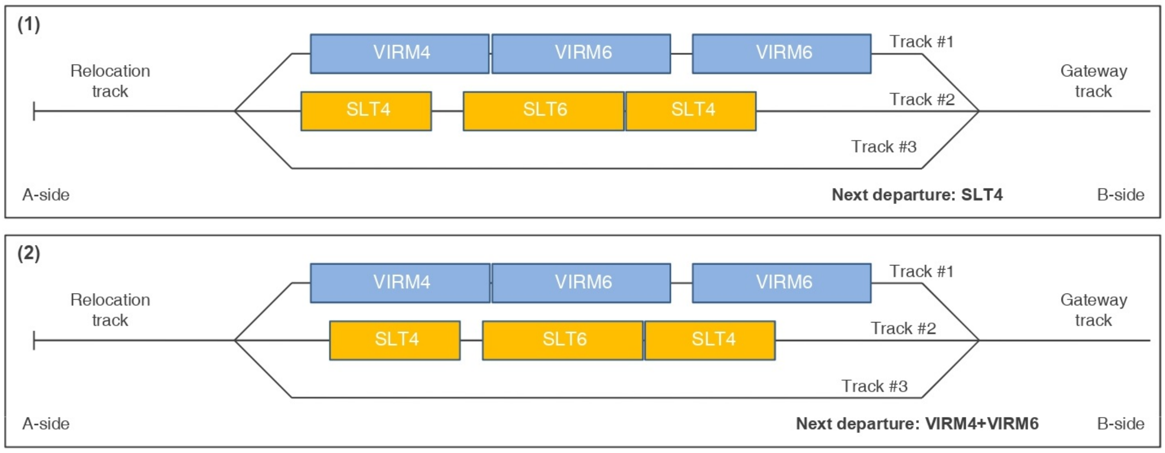

Figure 6.5: Examples of conflicting scenarios to illustrate the preconditions for conflict resolution in the TBS-based routing strategy.

3.a) Late combination: If among the fronts of all queues on the parking tracks $\tau \in T_p$ there are sufficient candidate train units of certain type and subtype such that can combine with each other and create the correct composition for the next departure, then the candidate train units will be rewarded to move individually (and sequentially) to the gateway track $\tau \in T_g$ right before its departure.

3.b) Relocation: The alternative to the late combination, which is not always possible, is to relocate a train parked in the back of a queue (on the relocation tracks side). To do so, the candidate train will be rewarded to move to a relocation track $\tau \in T_r$. The moving train may already be in the correct composition for the next departure. Otherwise, if there is no such composition in the front of the queues, individual train units such that can combine and create the correct composition will be prompted to move and combine on the relocation track, similarly as in **3.a)** on the gateway track. From the relocation track $\tau \in T_r$, the train in the correct composition for the next departure will be rewarded to move towards the gateway track $\tau \in T_g$ via a non-conflicting path for departure. The latter implies that there should always be at least one parking track $\tau \in T_p$ empty so that relocation movements are possible at any time.

An example of such event and how relocation can solve it is shown in Figure 6.6. In this case, the next departure is a train unit VIRM4, which can only be found in the back of one of the train queues. Therefore, after identifying the candidate train unit for the next departure, the unit will move to the relocation track where it will reverse direction (1) and then move towards the gateway track via a non-conflicting path, generally an empty parking track (2).

4) Unnecessary movements rules: Furthermore, in order to prevent unnecessary movements during the stay of a train unit in the shunting yard (i.e. beyond the strictly necessary movements towards service facilities, parking tracks and gateway tracks) in *Middle* and *EndService* triggers, and only if all the aforementioned conflict resolution rules have not been executed, we assign a negative reward to those trains with service tasks completed and already parked on a parking track $\tau \in T_p$ such that choose the action *Move*. This applies for train units that are in the correct composition for a departure and for train units that are still not in the correct composition when the train unit to be combined with is being serviced or has not arrived yet at the shunting yard. For those trains that have just completed a service task (i.e. event *EndService*), if the agent selects the *Move* action, we assign a negative reward if the destination track is a gateway track $\tau \in T_g$ or a relocation track

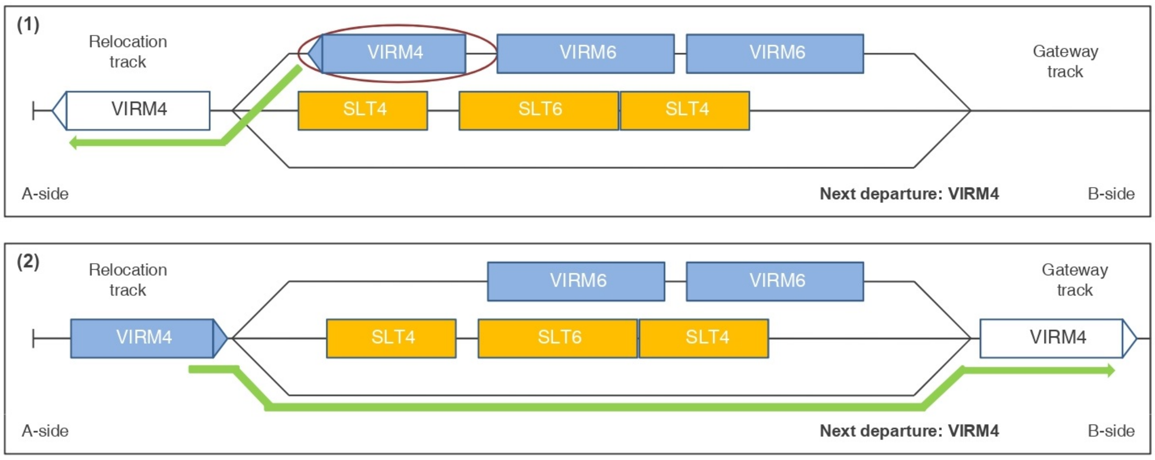

Figure 6.6: Representation of the resolution of a conflict by means of relocation. The train unit VIRM4 at the back of the queue is the only possible candidate for the next departure. It moves to the relocation track (1) and then to the gateway track via a non-conflicting path (2).

$\tau \in T_r$ and no reward if it is a parking track $\tau \in T_p$. Finally, in *PrepDeparture* and *Departure* triggers, if the train is not next to depart, has service tasks completed and is parked on a parking track $\tau \in T_p$, then it will get a negative reward if it selects the action *Move* and regardless of whether they are in a composition matching a scheduled departure or not.

6.2.3. In-Residence Time Strategy

The In-Residence Time Strategy, as described by Beerthuizen (2018) consists in assigning train units to parking tracks based on its departing time, creating a chronological departure order of trains on each track independently of the rolling stock types.

As explained in chapter 5, in the approach with 0% matching, match indexes change as trains depart from the shunting yard in a way that train units can suddenly find itself blocked by another agent. Consequently, the In-Residence Time Strategy does not fit with this design, as in this strategy we need to have the certainty of what exact train unit(s) will depart at what time in advance, so we can park them correctly. Therefore, for this parking strategy we define the procedure with train unit IDs specified for all departing trains (100% matching) and hence we already know what the position of each train unit in the departure sequence will be at the beginning of the instance. This assumption ensures that we can park train units in a chronological departure sequence order without provoking blockages during the process. Figure 6.7 gives an example of parking according to this strategy.

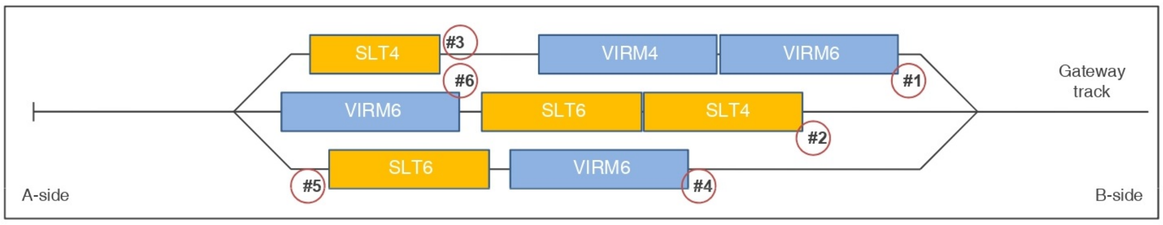

Figure 6.7: Example of the IRTS standard parking rules with trains of types VIRM and SLT. The position of each train in the departure sequence is shown next to each train.

In the TORS-MATDRL framework, the basic idea is to reward agents for parking trains respecting the order according to their position in the departure sequence. Therefore, with the IRTS parking rules, for all parking tracks, the departure indexes for the trains in the queue must be monotonic to ensure that they can leave the shunting yard according to the departure sequence without conflicts. We say that the IRTS parking rules are violated when the described parking order is violated on a parking track $\tau \in T_p$.

The IRTS-based Routing Strategy consists of the following four components:

1) Standard parking rules: In this routing strategy, in *Middle* and *EndService* event triggers, we reward those movements that ensure a non-conflicting departure sequence according to the scheduled departures, arriving from both the A-side and the B-side of the shunting yard towards a parking track $\tau \in T_p$. When this is violated, a negative reward is assigned. In order to include the combination subproblem in the framework, for those train units that are to be combined, the idea is to keep them parked on the edge of a parking queue in the front side (gateway track side) until the correct train unit ID to be combined arrives. Therefore, we assign a reward when the arriving train at the correct track (where the train waiting to be combined is parked) is the correct train unit ID to be combined with, and a negative reward otherwise. Furthermore, it should be noted that the rules described allow a train unit that is to be combined to park next to a train unit that does not need to be combined as long as the parking rule is not violated, and we punish such parking otherwise. Lastly, it should be noted that service tasks have priority over the standard parking rules. Therefore, train movements associated with servicing in dedicated service facilities are carried out regardless of the resulting queue of parked trains on the destination track, if applicable. By contrast, for those service tasks that can be carried out on any parking track, the standard parking rules still prevail.

2) Combination and split rules: in the IRTS-based routing strategy and according to the rewards presented in section 5.3, train units are rewarded for combining or splitting when the train units did not match any composition in the departure sequence before the operation but match one after the operation. Therefore, since these operations involve specific train units in its definition (defined by their IDs), then if there is need for combination in the scenario, then there is one and only one possible correct combination operation. Likewise, if there is need for splitting, then there is only one and only one possible correct splitting operation. Combination and splitting are allowed on any track $\tau \in T$.

3) Conflict resolution rules: During the attempt to solve an instance, certain situations often inevitable can happen in which the strategies presented so far are insufficient to drive agents to do the correct sequence of actions to solve it. For instance, trains to be combined might end up parked on different tracks, or standard parking rules might be violated. Therefore, we introduce the following strategies for conflict resolution: (a) late combination on gateway track and (b) relocation.

3.a) Late combination on gateway track: If a train unit is the next to depart but is still not in the correct composition, and it is in the front of the queue (gateway side), then we prompt to move it to the gateway track $\tau \in T_g$ up to three triggers before the *Departure* trigger, which gives an additional chance to these train units to solve the combination subproblem. In this case, the second train unit of the departing train can also move from a different parking track $\tau \in T_p$ to the gateway track $\tau \in T_g$ following the same rule or it can also be the case that it is already waiting on the gateway track. In this way, the combination of both trains is possible on the gateway track $\tau \in T_g$ right before the departure. Note that in this case, the standard parking rules were not violated.

3.b) Relocation: When a large enough number of train units are being handled at the shunting yard, blocking of train units may occur by inevitably violating the standard parking rules. In particular, when the number of train units parked exceeds the number of available parking tracks, the chances that arriving train units will inevitably violate the parking strategy significantly increase. In such cases, when the parking rules are violated, relocation of train units is needed. In other words, the violation of the standard parking rules is the key precondition to trigger a relocation movement. A relocation movement consists in moving certain train units towards a relocation track $\tau \in T_r$ and back to a parking track $\tau \in T_p$ in order to restore the non-conflicting parking order in the shunting yard so that trains can depart according to the scheduled departure sequence without blocking each other.

An example of such event and how relocation can solve it is shown in Figure 6.8. The match numbers indicate the position of each train in the departure sequence, i.e. *match = 1* is the next to depart, while *match = 4* is the last to depart. In (1), an arriving train unit with match equal to 4 inevitably blocks another train unit. In (2), the train unit with match equal to 3 moves towards the relocation track and then in (3) it moves to a different parking track. The resulting parking queues are no longer conflicting and hence trains can depart. It should be noted that it is not the only possible solution. Nonetheless, if the train unit with match equal to 4 initially moved next to the train unit with match equal to 1, more movements would be needed, assuming that the relocation track is long enough to accommodate more than one train unit.

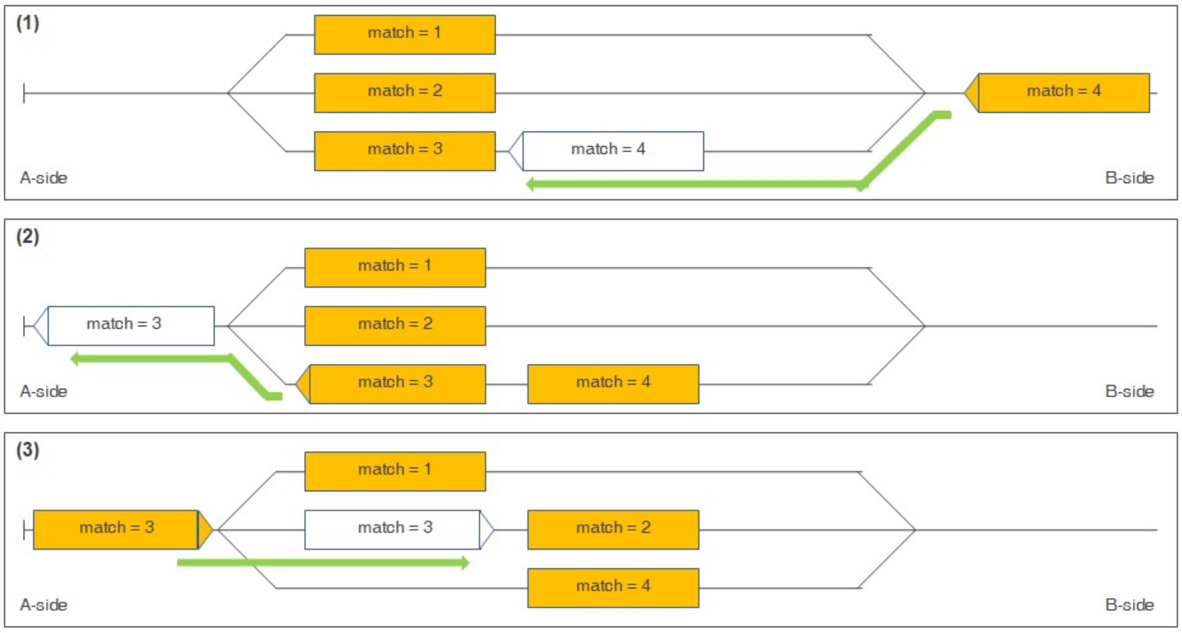

Figure 6.8: Representation of a state that creates a conflicting departure sequence (1) and hence it creates the need for relocation, and how relocation can solve the conflict (2-3).

In particular, in the TORS-MATDRL framework, when the departure sequence on a certain parking track $\tau \in T_p$ is conflicting, we reward the train unit parked in the back of the queue (i.e. A-side or relocation track side) to move towards a relocation track. This procedure is repeated until there is no violation anymore, i.e. the departure sequence on the parking track is no longer conflicting. Trains that moved to the relocation track $\tau \in T_r$ will be given again the chance to perform a correct parking according to the conditions described for standard parking, combination and splitting, and hence to restore the non-conflicting departure sequences throughout all parking tracks $\tau \in T_p$ in the shunting yard.

4) Unnecessary movements rules: Furthermore, in order to prevent unnecessary movements during the stay of a train unit in the shunting yard (i.e. beyond the strictly necessary movements towards service facilities, parking tracks and gateway tracks) in *Middle* triggers, and only if all the aforementioned conflict resolution rules have not been executed, we assign a negative reward to those trains with service tasks completed and already parked on a parking track $\tau \in T_p$ such that choose the action *Move*. This applies for train units that are in the correct composition for a departure and for train units that are still not in the correct composition when the train unit to be combined with is being serviced or has not arrived yet at the shunting yard. For those trains that have just completed a service task (i.e. event *EndService*), if the agent selects the *Move* action, we assign a negative reward if the destination track is a gateway track $\tau \in T_g$ or a relocation track $\tau \in T_r$ and no reward if it is a parking track $\tau \in T_p$. Finally, in *PrepDeparture* and *Departure* triggers, if the train unit is not the next to depart, in the correct composition, with service tasks completed and parked on a parking track $\tau \in T_p$, then it will get a negative reward if it selects the action *Move*.

6.2.4. Overview of the Routing Strategies and Limitations of the Implementations

Both the TBS and IRTS-based routing strategies described in the previous sub sections have certain limitations. These include, for both of them, the following:

- Only one gateway track is available, which simplifies routing.
- Trains are assumed to consist of maximum two train units, although there are train series operated by NS, such as the ICM, which often run in compositions consisting of three of such train units. The implementations are designed to solve combination and splitting for trains consisting of maximum two train units and do not work for longer compositions, for which extensions are necessary.
- Conflict resolution strategies target the most common conflicts but there are many others less likely to happen that cannot be solved with the described strategies.
- Conflict resolution strategies in the TBS-based routing strategy are not suited to solve conflicts involving train units blocked on both sides of a parking track.

Finally, Table 6.2 shows an overview of the main characteristics of both the TBS-based routing strategy and the IRTS-based routing strategy as described in subsection 6.2.2 and subsection 6.2.3.

	TBS-based Routing Strategy	**IRTS-based Routing Strategy**
Matching Subproblem	Departures defined by train unit types and subtypes required (i.e. not solved)	Departures defined by train unit IDs required (i.e. solved)
Uncertainties	Arriving and departing train units	Arriving train units
Standard Parking	Stacking trains of the same type on the same track	Trains ordered on parking tracks according to departure sequence
Combination and Splitting	Strategy meant to manage numerous possibilities	Unique correct possibilities
Conflict Resolution	(1) Late combination on gateway track (2) Relocation: relocate candidate train or combination on relocation track	(1) Late combination on gateway track (2) Relocation: restoration of chronological parking order
Unnecessary Movements	Trains parked, with service tasks completed and no standard parking rules violation wait on its current position until (shortly before) its departure time	
Application	Shunting yard capacity assessment	Planning of daily operations

Table 6.2: Overview of the TBS and IRTS-based routing strategies.

6.3. Answer to Sub-Question 2

The development of the routing optimization solution described in this chapter allows us to answer the second research sub question: **How can routing strategies be formulated in the multi-agent deep reinforcement learning framework in order to produce optimal route plans and speed up learning?**

The routing optimization solution consists of the Heuristic for Random Exploration and the Heuristics for Routing Strategies, which include a TBS-based routing strategy and an IRTS-based routing strategy. Firstly, the heuristic for random exploration based on the principles of large neighbourhood search is required to efficiently search the action space, in which the distribution and importance of actions is highly imbalanced. Additionally, this heuristic significantly helps speeding up the learning process, which by pure random exploration would be computationally very expensive.

Secondly, two routing strategies are developed and implemented based on the parking strategies introduced by Beerthuizen (2018), which are the Type-Based Strategy (TBS) and the In-Residence Time Strategy (IRTS) respectively, which we extend in this work to include complete routing strategies consisting of four components: (1) standard parking rules, (2) combination and split rules, (3) conflict resolution rules and (4) unnecessary movements rules. The routing strategies are formulated as reward functions in the TORS-MATDRL framework. The resulting routing strategies are named after the parking strategies, i.e. TBS and IRTS-based routing strategies. The reward functions try to give rewards as much as possible and are carefully shaped to ensure a hierarchy in the sequence of decisions and events that agents have to make to complete an instance correctly. Lastly, the routing strategies are designed in a way so that they are scalable but have certain limitations in their implementations.

7 Adaptations and Formulation of Routing Strategies

In this chapter we present the adaptations and formulations of the routing strategies developed in chapter 6. It is important to note that the routing strategies developed in section 6.2 are described for a generic shunting yard track layout. Nevertheless, although it is often possible to identify some types of shunting yard track layouts such as the shuffleboard or *carrousel* types, as described in section 3.2, most shunting yards may have many layout particularities which can make them quite unique in practice. Therefore, the TBS and IRTS-based routing strategies developed need to be adapted to fit particular problem designs. Firstly, we present an overview of the case studies that will be used for our experiments (section 7.1), and in the successive sections we describe the adaptations of the routing strategies for the *carrousel* type of shunting yard without train flow pattern constraints (section 7.2), for the *carrousel* type of shunting yard with train flow pattern constraints (section 7.3) and for the shuffleboard type of shunting yard (section 7.4).

7.1. Overview of Case Studies

We carry out our experiments in two case studies, which are both *carrousel* types of shunting yards, one of them without train flow pattern constraints and another one with flow pattern constraints. 2 real-life shunting yards will be addressed for this, which are the shunting yards of Heerlen and Kleine Binckhorst, both owned and operated by NS, as examples of each of the shunting yard types considered. The reason to use two case studies with different infrastructure layouts and different operating modes is to draw insights on how the routing strategies have to be designed or adapted in order to fit specific problems, as well as to identify and underpin the commonalities and differences between those.

On one hand, Heerlen is a small node with lower complexity of planning assignments compared to other nodes such as Eindhoven (Baurichter, 2020). The track layout of Heerlen is shown in Figure 7.1, where the station is shown on the right side with two platforms, and the shunting yard is located on the top centre and represented by dashed-line tracks. Trains usually arrive from the A-side side to platform 2, where they will reverse their direction and they may access the shunting yard in the opposite direction of arrival. The shunting yard can be accessed via track 213 at signal 54, which connects directly to platform 2 and to the mainline with track 205. Heerlen consists of a range of parallel free tracks (31 to 34; the rest of parallel tracks are disregarded in the case study for simplicity), which can all be accessed from both sides, and a relocation track on the left side (39), which can only be accessed from the right side, and hence it is a LIFO track. Due to this infrastructure layout, parking tracks as used in both directions. Lastly, there is a cleaning platform serving part of tracks 33 and 34, where the associated service tasks can be carried out.

Janssens (2017) classifies this shunting yard as a station type. Nevertheless, in our simplification of Heerlen all tracks are open on both sides and hence it can be considered as a *carrousel* type, although trains do not really have to move around the shunting yard as a general rule since the only service facility is located on two of the parking tracks. Therefore, we classify this shunting yard as a *carrousel* type without train flow pattern constraints.

Besides, one characteristic feature of the node of Heerlen is the presence of *non-service traffic*. The department of Research & Development Node Logistics at NS understands *non-service traffic* as the on-going train traffic traversing the shunting yard carrying passenger traffic, which need no servicing by contrast to *service traffic*. Furthermore, all non-service traffic have a defined itinerary within the shunting yard and arrival and departure times. Therefore, planning non-conflicting routes between service traffic and non-service traffic within the node is part of the planning assignments at the node of Heerlen.

On the other hand, Kleine Binckhorst is a shunting yard located very close to The Hague Central (terminus) station and next to the mainline towards Zoetermeer and Gouda. Figure 7.2 the infrastructure layout in the area, with the mainline and infrastructure owned by ProRail represented with continuous black lines while the infrastructure of the shunting yard, owned by NS is represented by dashed lines. In Kleine Binckhorst, the shunting yard can be accessed via signal 76 on the bottom left and via signal 434 on the right side, and all parking tracks (52 to 59) can be accessed from both sides. There are service facilities for inspection (64), internal cleaning (61 and 62) and external cleaning (63), all located on the right side of the parking tracks. For simplicity, in our Kleine Binckhorst case studies we only consider internal cleaning as the set of service tasks that can be performed, which can only be carried out on tracks 61 and 62. Additionally, track 63 also functions as a relocation track. Lastly, Kleine Binckhorst operates as an isolated node, in the sense that there is no non-service traffic, by contrast to the node of Heerlen.

Kleine Binckhorst is a good example of a *carrousel* type of shunting yard with flow pattern constraints in which trains move around the yard along different service locations. Therefore, the way this shunting yard is operated by NS differs significantly from the operation of Heerlen. Assuming a single entry track to the shunting yard on signal 76, all trains arrive via that signal and are directed towards the entry tracks 56 to 59, on the north side, from which trains are moved to the service facilities located on track 11 and 12. From there, trains are directed towards the parking tracks 52 to 55 on the southern side via relocation track 13, where trains change direction. On tracks 52 to 55, trains wait until their scheduled departure time and leave the shunting yard using the same track used for entry. Therefore, all tracks except of the relocation track are used in one single direction. In particular, tracks on the northern side (56 to 59 and 61 to 62) are used towards the B-side and tracks on the southern side are used towards the A-side. As a result, trains describe a circular flow around the shunting yard. This flow pattern applies for all trains regardless of the service tasks that they have assigned (i.e. trains without service tasks follow the same pattern). Figure 7.3 visualizes the described flows in the yard.

Finally, Table 7.1 presents an overview of the differences and commonalities between the shunting yards of Heerlen and Kleine Binckhorst. In conclusion, although both case studies share some characteristics such as the fact that all parking tracks are open on both sides, the way they are operated is significantly different and hence we can get insightful conclusions that can support our answer to the third sub question: **How flexible and scalable are the routing strategies in different shunting yards?**

	Heerlen	**Kleine Binckhorst**
Type	*Carrousel* shunting yard without flow pattern constraints	*Carrousel* shunting yard with flow pattern constraints
Track usage	Bidirectional	Single direction
Parking tracks	4 free tracks	4 free tracks
Entry tracks	No	4 free tracks
Exit tracks	No	No
Relocation tracks	1 LIFO track	1 LIFO track
Service facilities	Two free tracks directly connected to two parking tracks	Two "drive-through" (free) tracks
Critical tracks	Relocation track	Relocation track

Table 7.1: Overview of differences and commonalities between the shunting yards of Heerlen and Kleine Binckhorst.

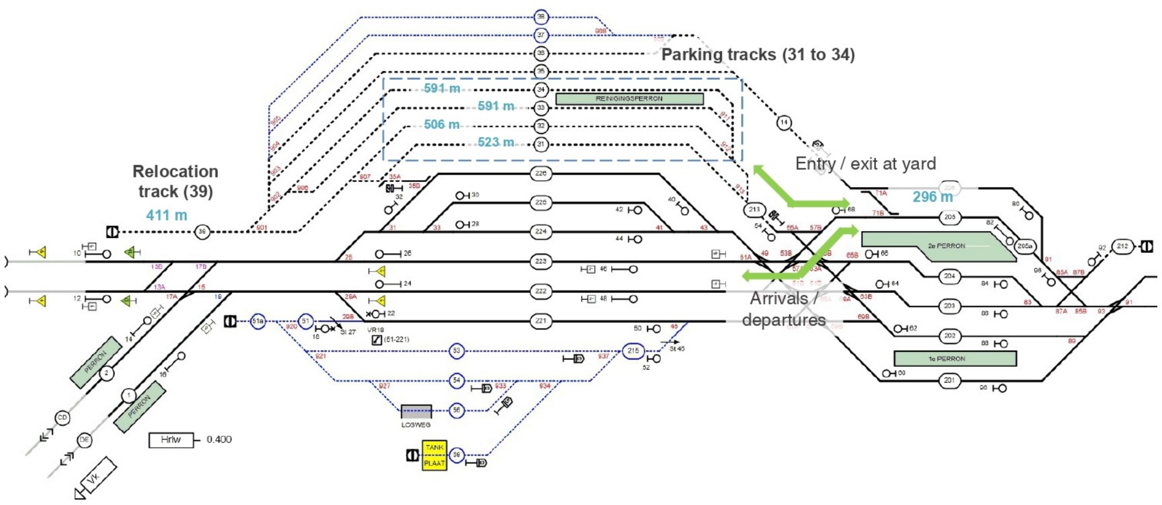

Figure 7.1: Heerlen shunting yard (top centre, dashed lines consisting of tracks 31 to 39) and station (right) (Sporenplan Online, 2019).

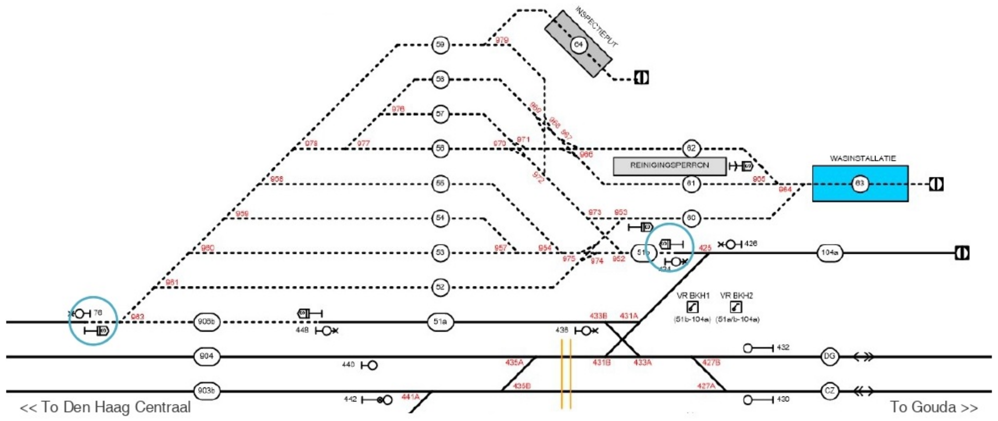

Figure 7.2: Kleine Binckhorst shunting yard (Sporenplan Online, 2019). The entry tracks are marked with blue circles.

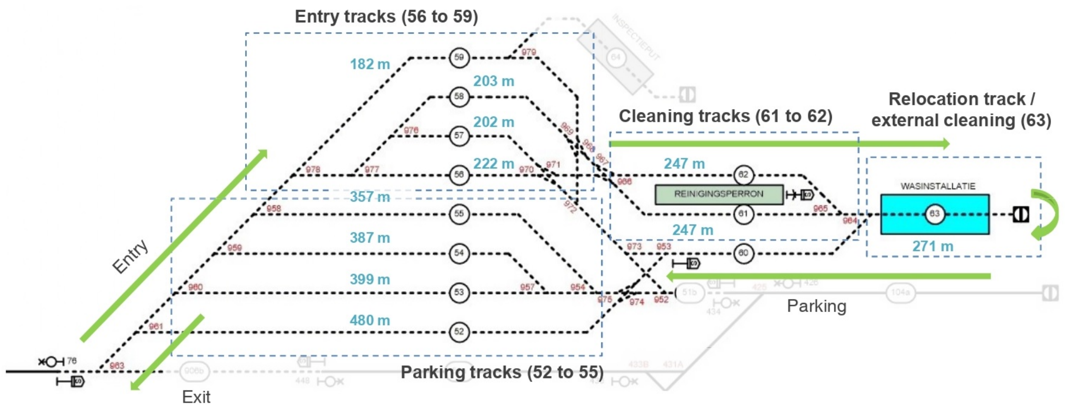

Figure 7.3: Kleine Binckhorst shunting yard with operation details (adapted from Sporenplan Online, 2019).

7.2. Adaptations of the Routing Strategies for Carrousel Shunting Yards without Flow Pattern Constraints

In the *carrousel* shunting yards without flow pattern constraints there are no particular constraints in the flow of trains moving around the shunting yard. Therefore, the routing strategies described for both TBS and IRTS described in section 6.2 can mostly be applied directly to this shunting yard. However, when we apply the routing strategies in the specific case of Heerlen, its particularity of having the service facility serving part of two of the parking tracks has to be taken into account. To do so, we prompt agents to clear the service platform by repositioning on the same track (on the part without service platform) and hence allowing other trains to use the service platform. After servicing, if the standard parking rules are violated, trains will follow the conflict resolution rules as described.

In particular, relocation is extended with the following: for those trains that have just completed a service task (i.e. event *EndService*), the following applies: if the train unit is not in the correct composition, it is blocked on the gateway track side and chooses a *Move* action from parking track $\tau \in T_p$ to relocation track $\tau \in T_r$ and the train unit to be combined with is either on a different track $\tau \in T_p$ or it has not arrived yet at the shunting yard, then the former receives a positive reward. This relocation movement will increase the chances to find the train unit to combine with significantly.

The complete reward functions are presented in Appendix B and all build on Algorithm 1 for the original assignment of rewards, presented in section 5.3. For the new rewards, we introduce the parameter *Reward.Park*, with value equal to 0.5. On one hand, Algorithm 3 shows the formulation in the TORS-MATDRL framework of the procedure for the TBS-based routing strategy in Heerlen. On the other hand, Algorithm 4 shows the formulation in the TORS-MATDRL framework of the procedure for the IRTS-based routing strategy in Heerlen.

7.3. Adaptations of the Routing Strategies for Carrousel Shunting Yards with Flow Pattern Constraints

By contrast to the previous, in this case we have constraints on the flow of trains moving around the shunting yard as described previously. In the particular case of Kleine Binckhorst, the flow of trains follows the following pattern (please refer to section 4.1 for further details on notation for infrastructure modelling): (1) trains arrive via the gateway track $\tau \in T_g$, (2) from where they move to an entry track $\tau \in T_e$, (3) then to a service facility $\tau \in T_s$ and (4) from there to a relocation track $\tau \in T_r$. Finally, trains change direction and (5) move to the parking tracks $\tau \in T_p$, from where (6) they will move to the gateway track $\tau \in T_g$ for departure. In order to guide agents to follow this pattern, it is necessary to adjust the rewards for moving and servicing accordingly and to extend the reward function with penalties to prevent the violation of the flow pattern. Nonetheless, in order to maximize the parking capacity of the shunting yard, we can allow trains to break the flow pattern and use tracks $\tau \in T_e$ for parking as well. Furthermore, the relocation track $\tau \in T_r$ is generally used to split trains. The latter and other relevant considerations to fit the design are listed as follows:

- In principle, standard parking rules apply in on parking tracks $\tau \in T_p$. The entry tracks $\tau \in T_e$ can be used as well for parking when arriving train units anticipate that will not be able to park correctly on parking tracks $\tau \in T_p$ according to the standard parking rules. In such case we exceptionally allow trains to violate the train flow pattern constraint and park on the entry tracks $\tau \in T_e$, and depart directly from there without using the relocation track $\tau \in T_r$. It is also often the case that the train unit that will not be able to park on $\tau \in T_p$ anymore will still have to be serviced on $\tau \in T_s$. In such case, the train will reverse on the service track and move back to the entry tracks $\tau \in T_e$ for parking.
- Train movements associated with service tasks of the same train unit must respect the train flow pattern of the shunting yard. Therefore, once the train moves to the relocation tracks $\tau \in T_r$ or to the parking tracks $\tau \in T_p$, the service tasks assigned on dedicated facilities on tracks $\tau \in T_s$ must have been already completed.
- Combination is only allowed on the parking tracks $\tau \in T_p$.
- Splitting is in principle allowed only on the relocation track $\tau \in T_r$. In TBS it is still however allowed to split on a parking track $\tau \in T_p$ whenever it is necessary to correct a composition for the next departure.
- The predefined train flow pattern is enforced and its violation is prevented except for each arriving train that anticipates that cannot park correctly on $\tau \in T_p$ anymore.
- Regarding conflict resolution strategies: On one hand, in TBS, relocation is triggered and executed under the same conditions as described in subsection 6.2.2. On the other hand, in IRTS, the underlying

idea for relocation is the same but the conditions under which it is triggered are different. The precondition for relocation is that on all parking tracks $\tau \in T_p$ there is at least one train parked, which departs later than an arriving train waiting on the relocation track $\in T_r$. It is hence necessary to decide what parking track $\in T_p$ to clear, and relocate all train units parked on it, probably one by one due to the limited length of the relocation track $\in T_r$. This process is quite elaborate and hence it is not implemented in TORS-MATDRL. We limit the conflict resolution strategy to letting trains violate the flow pattern when they cannot park correctly on $\tau \in T_p$ anymore as already described, and hence to park on $\tau \in T_e$ or $\tau \in T_s$ instead and depart from there.

- The strategy for conflict resolution consisting in late combination in the gateway track is not used either since due to the characteristic flow pattern in Kleine Binckhorst, using parking tracks in one single direction, the chances of having the two train units to be combined parked on different tracks and at the front of the queue (gateway track side) are much lower than in locations such as Heerlen, where the entry track to the parking tracks $\tau \in T_p$ is the same as the exit track.

The complete reward functions are presented in Appendix B and again all build on Algorithm 1 for the original assignment of rewards, presented in section 5.3. For the new rewards, we introduce the parameter *Reward.Park*, with value equal to 0.5. On one hand, Algorithm 5 shows the formulation in the TORS-MATDRL framework of the procedure for the TBS-based routing strategy in Kleine Binckhorst. On the other hand, Algorithm 6 shows the formulation in the TORS-MATDRL framework of the procedure for the IRTS-based routing strategy in Kleine Binckhorst.

7.4. Adaptations of the Routing Strategies for Shuffleboard Shunting Yard Types

The assumptions for the routing strategies described in section 6.2 need to be adjusted to fit the design for a shuffleboard shunting yard types (Figure 7.4). Although in this research we do not analyze this type of shunting yard type at the same level of detail as *carrousel* shunting yard types, we can still draw some guidelines on how applicable are the routing strategies are for such cases.

Using the convention presented in Figure 6.3, the main difference in shuffleboard shunting yards, compared with *carrousel* shunting yards is that the parking tracks $\tau \in T_p$ are LIFO tracks (i.e. open on one single side) and hence there is no possible relocation track $\tau \in T_r$ on the opposite site of the gateway track. Nevertheless, it is still possible to have relocation tracks $\tau \in T_r$ on the same side of the gateway side $\tau \in T_g$, or it can even be the case that both tracks overlap. Figure 7.4 shows an example of such a shunting yard.

From the routing strategies point of view, the adaptations needed are as follows:

- 1) Standard parking rules: these are applied as normally on each parking track $\tau \in T_p$. Nevertheless, in IRTS is should be noted that these rules can only (physically) apply when a train arrives from the B-side (gateway track side), since parking tracks $\tau \in T_p$ are LIFO tracks.
- 2) Combination and split rules: these are applied as normally.
- 3) Conflict resolution rules: late combination on gateway track can still be applied following the same procedures. Nevertheless, relocation needs adjustment in both TBS and IRTS.
  - **(1) TBS:** Given the preconditions described (which remain the same), if the candidate train(s) for the next departure are blocked on parking track $\tau \in T_p$, then it is necessary to relocate all the trains parked on the B-side (gateway track side) of the candidate train by moving them away to another track on the B-side so that the latter is no longer blocked on that side. The trains that were originally blocking the candidate train may be moved to a relocation track $\tau \in T_r$ different from the gateway track $\tau \in T_g$ so that the candidate train can depart without conflicts. In case the relocation track $\tau \in T_r$ and the gateway track $\tau \in T_g$ happen to be the same track, then the trains being relocated will need to move again to a different parking track $\tau \in T_p$ according to the standard parking rules. In conclusion, the key difference in shuffleboard type of shunting yard compared to *carrousel* shunting yards is that in the latter, the train relocating is the candidate train, whereas in the former, the trains relocating are those that block the candidate train.

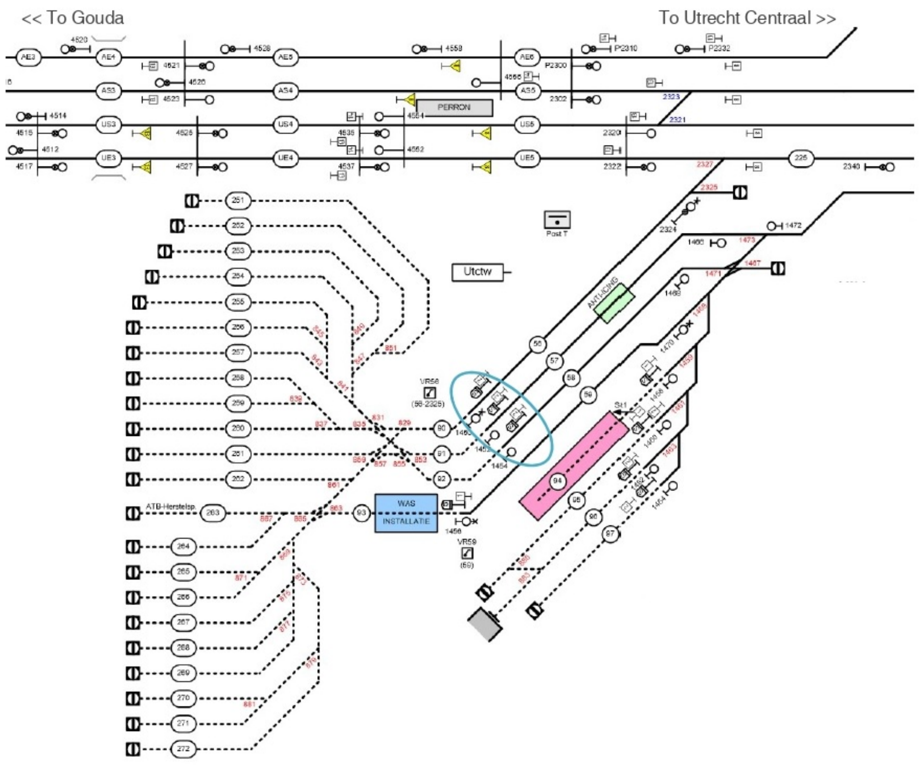

Figure 7.4: Cartesuisweg shunting yard (Sporenplan Online, 2019). The entry tracks are marked with blue circles.

- **(2) IRTS:** Relocation is solved following the exact same procedure as in TBS for shuffleboard shunting yards. Again, the next train to depart (in this case unique candidate train) does not relocate and only the trains blocking it will do so.

- 4) Unnecessary movements rules: these are also applied as normally.

7.5. Answer to Sub-Question 3

The adaptation and formulation of the routing strategies developed to different case studies as described in this Section allows us to answer the third research sub question: **How flexible and scalable are the routing strategies in different shunting yards?** In chapter 6 we have developed on one hand an heuristic for random exploration, which are deemed to be applicable for any kinds of shunting yards since these focus on particular sequences of actions that we want to stimulate agents to learn, such as setting back instead of waiting, clearing the gateway track, departing, starting a service task if there are pending service tasks, or performing a correct combination or split.

On the other hand, we have developed the TBS and IRTS-based routing strategies for a generic shunting yard consisting of a gateway track $\tau \in T_g$, a set of parallel parking tracks T_p, and a relocation track $\tau \in T_r$ on the opposite side of the gateway track, in such a way that all parking tracks can be reached from both the gateway track and the relocation track without reversing the train direction. Each routing strategy consists of four components, which are (1) standard parking rules, (2) combination and split rules, (3) conflict resolution rules and (4) unnecessary movements rules. These rules can be applied at any type of shunting yard but need specific adjustments to fit each type (e.g. *carrousel*, shuffleboard), and further adjustments to fit the particular characteristics of real-life shunting yards.

Firstly, for the *carrousel* type of shunting yard without flow pattern constraints, the routing strategies as described in section 6.2 can be applied directly to such case. Secondly, for the case of *carrousel* shunting yards with flow pattern constraints, extensions to stimulate agents to learn to follow a predefined flow pattern are required, as well as further extensions to let agents violate this flow in particular cases for conflict resolution.

For this, a distinction between groups of parking tracks can be made, possibly including entry tracks T_e or exit tracks T_x on top of normal parking tracks T_p. Entry or exit tracks may be used for temporary parking or sorting, whereas normal parking tracks are used preferably for long parking and train sorting. Also, when parking tracks are used in single directions it means that in general, most trains will have to reverse their direction on a relocation track $\tau \in T_r$ in order to close the characteristic circle of *carrousel* shunting yards. The relocation track might therefore become a bottleneck and hence we may prefer to speed up all activities carried out upstream or move them downstream as much as possible, such as splitting and combination operations. Therefore, the environment in this case is generally more constrained compared to the case without such flow constraints. In terms of conflict resolution, since parking tracks are used in one single direction, the preconditions for conflict resolution are also different compared to the case without flow constraints, which slightly modifies for instance the way relocation is handled.

The real-life shunting yards considered for the experiments in this research have further particularities that have to be taken into account in the adaptations of the routing strategies, such as the location of service facilities and how these are connected to parking tracks and other tracks.

Lastly, further adaptations of the routing strategies as described in section 6.2 are needed for the shuffleboard type of shunting yards and for hybrid or other unclassified shunting yards. The main difference in shuffleboard shunting yards is that parking tracks $\tau \in T_p$ are LIFO tracks. The standard parking rules are applied as normally (but only from the open side of parking tracks), as well as combination and split rules and unnecessary movements rules. Nonetheless, conflict resolution rules again need adaptations to fit this particular problem design. The preconditions to trigger relocation movements stay the same but the way train movements are handled is quite different since only the gateway track side of the parking tracks $\tau \in T_p$ is open.

In conclusion, the routing strategies described in section 6.2 are flexible enough to be applied to various types of shunting yards. The fundamental four components of each routing strategy are generally replicable to various contexts and further extensions and constraints may be added to fit particular problem designs. Lastly, we have shown how this can be formulated in a multi-agent DRL framework.

8 Experiments

In this chapter we present the experiments carried out. Firstly, we present an overview of our objectives and the experiments carried out to achieve such objectives (section 8.1). Secondly, we introduce the general experiments setup used (section 8.2) where we describe the data generation process, objectives and performance metrics, benchmarking, learning parameters, action triggers and violations. Following this, we present Experiments 1 to 4 (section 8.3 to section 8.6), the discussion of the results in section 8.7 and lastly the answer to sub-question 4 in section 8.8.

8.1. Overview of Objectives and Experiments

The main objective of the experiments is to demonstrate the capabilities of the heuristic for random exploration and the TBS and IRTS-based routing strategies developed in the TORS-MATDRL framework to solve the TUSP with splitting, combination and servicing in different contexts. In particular, we want to explore the performance of our implementations in different problem designs, defined mainly by the predefined train matching and the type of shunting yard. We carry out our experiments in two shunting yard types, each with different track layouts. We consider a *carrousel* type of shunting yard without train flow pattern constraints (Figure 7.1) and a *carrousel* type of shunting yard with train flow pattern constraints (Figure 7.3). In essence, the existence of flow patterns is caused by specified groups of tracks for entry, parking and exiting. By contrast, in the case without such constraints, there is no distinction of track types for either entry, parking or exiting. The heuristic for random exploration is included in all TORS-MATDRL trainings and service tasks are included in all experiments. The performance is measured on the basis of the computational results and our performance metrics (defined in subsection 8.2.2). The results are bench marked against HIP (see section 3.2).

We carried out a total of four experiments. Firstly, we assess the performance of both the TBS-based routing strategy and the IRTS-based routing strategy by benchmarking them against the original settings (baseline variant) in TORS-MATDRL (NS, 2020) on the basis of the computational results and our performance metrics. By original settings we mean (1) without the routing strategies and (2) with the original reward functions (Algorithm 1). Based on our experience, with the original settings it is quite difficult for the agents to learn to solve instances and for the algorithm to converge. Therefore, in order to speed up the learning process and get meaningful results to compare with, we include our heuristic for random exploration in this baseline variant. To carry out this assessment, we use simpler problems in a *carrousel* shunting yard without train flow pattern constraints.

The focus of the second experiment is on analyzing the performance of our routing strategies to solve the matching and combination subproblems. On one hand, with no predefined train matching (i.e. departures consist of lists of requested rolling stock types and subtypes) it is particularly interesting to analyze how agents solve these subproblems in the TBS-based routing strategy. Therefore, for these experiments we will use representative instances in which agents have to solve several matching and combination subproblems, as well as service tasks. For simplicity, we include one single rolling stock type. The key characteristics of such instances is that for each train departure there are several possible train unit candidates that can be part of the requested composition and hence our goal is to analyze how agents solve the associated subproblems to solve the complete instance correctly. On the other hand, we use the IRTS-based routing strategy to solve the same instances but with predefined matching (i.e. each departure consists of exact train unit IDs required). This will allow us to gain insights on the performance of each routing strategy to solve comparable instances on the basis of the computational results and our performance metrics. We carry out this experiment in the case study of a *carrousel* shunting yard without train flow pattern constraints (i.e. the Heerlen simplified shunting yard). Lastly, we benchmark both approaches against HIP. Therefore, we present four sets of results in total.

The focus of the third experiment is similar to the first experiment but we introduce multiple rolling stock types, which makes the problem more complex and realistic. We carry out this experiment in the same shunting yard as well. Our goal is therefore to evaluate the effects of having multiple rolling stock types in the shunting yard as an additional constraint to solve the matching and combination subproblems. We also use both the TBS-based routing strategy (for no matching) and the IRTS-based routing strategy (with predefined matching) and we benchmark both approaches against HIP.

Fourthly, we extend the previous experiments on matching, combination and servicing to include the splitting subproblem. The goal is to explore the effects of having both combination and splitting subproblems in the same instance, making the problem more realistic. Additionally, we carry out this experiment in a *carrousel* type of shunting yard with train flow pattern constraints (i.e. Kleine Binckhorst) to explore the operation of (groups of) tracks in one single direction. We also use both the TBS-based routing strategy (for no matching) and the IRTS-based routing strategy (with predefined matching) and we benchmark both approaches against HIP, reporting four sets of results in total.

The experiments are carried out using an Intel Core i7-7500U CPU with 16 GB RAM.

8.2. Experiments Setup

In this section we describe the data generation process, the objectives and performance metrics, benchmarking, learning parameters, events triggers and violations.

8.2.1. Data Generation

For our TORS-MATDRL framework we need a sufficiently large set of problem instances containing sequences of train arrivals and departures to be used during training and testing. One of the key aspects of machine learning in general is the concept of generalization, by which the model is able to solve unseen problem instances. Therefore, the set of training instances has to be sufficiently large and diverse in terms of parameters in order to ensure a proper learning process.

For this, we use an Instance Generator developed by NS (NS, 2020), which is in essence an optimization programme based on Mixed Integer Programming (MIP). The Instance Generator ensures (1) flow conservation between the incoming and outgoing flow of train units in the shunting yard and (2) compatibility of the sequences in time, such that all arrival events occur before all departure events and with sufficient time for movements and servicing (see subsection 4.2.1). Therefore, both conditions ensure that the problem is solvable. As a result of these conditions, the number of incoming train units and carriages equals the maximum number of train units and carriages that will be present in the shunting yard, which must be lower than the available parking space on the parking tracks. Furthermore, the number of trains arriving and departing at the shunting yard determines the problem complexity.

The inputs of the Instance Generator are the location, the train unit types considered and their characteristics and parameters such as the arrival and departure ratios of each possible train composition and arrival and departure time distributions, which have been adjusted to fit particular problem designs in terms of train compositions arriving and departing at the shunting yard, and the available service tasks and their frequency for each train unit type and subtype. Service tasks that can be carried out on any parking track have been removed for simplicity under the assumption that these generally do not really have an influence on routing and we consider only those that need to be carried out at dedicated facilities. Table 8.1 shows the service tasks used in the case studies for Heerlen and Kleine Binckhorst, with the details for each train unit type and subtype. These service tasks are assigned to each train unit with the probability associated with its frequency.

Service Task	**Train unit**	SLT4	SLT6	VIRM4	VIRM6
Repair (Heerlen)	Frequency	5 days			
	Duration [min]	120	120	120	120
Internal cleaning (Kleine Binckhorst)	Frequency	Daily			
	Duration [min]	14	17	11	14

Table 8.1: Overview of service tasks details used in the case studies in Heerlen.

The output is a set of instances consisting of a set of arrival and departure events, each with a train composition and event time associated, as well as the service tasks assigned for each train unit.

In order to randomize the instances, we vary the number of arriving trains and the random seed of the Instance Generator, which generates different train arriving and departure sequences, although these follow the patterns governed by the arrival and departure time distributions. The number of arriving trains should be large enough to obtain representative instances for each case study. For instance, we expect to see multiple trains parked on the same parking tracks in order to test our parking strategies. Nevertheless, it should also be noted that a higher number of arriving trains will also result in a higher complexity and thus training time till the convergence of the algorithm, which has to be bounded due to the time constraints of this research. Therefore, it is necessary to trade off between the representativeness of the instances in order to get meaningful results and the computational time.

Besides, it is important to note that we only have realistic parameters for the Instance Generator available for Kleine Binckhorst, in terms of rolling stock types and subtypes, arrival and departure ratios and arrival and departure time distributions. Therefore, we will use the same input parameters to generate instances in both Kleine Binckhorst and Heerlen. Consequently, the instances that we will generate for Heerlen will not be realistic since they will differ from the real scenarios with different train unit types using the shunting yards, arrival and departure ratios, service tasks carried out, etc. Nevertheless, the generated instances will still be useful enough to carry out our experiments in Heerlen. It should also be noted that the Instance Generator is not designed to generate non-service traffic. Therefore, this a limitation for our experiments in the node of Heerlen as our instances will not feature non-service traffic. This could however be included in TORS by simply blocking temporarily the route path used by non-service traffic during the time it is being used.

In our case studies we consider train units of types *Verlengd InterRegio Materieel* (VIRM), which stands for lengthened interregional rolling stock, and *Sprinter Lighttrain* (SLT) of the Dutch national railways NS. VIRM trains are used for Intercity services, whereas SLT trains operate Sprinter train service for commuter services. On one hand, VIRM trains can be of subtype '4' or '6', which indicates the number of carriages of in the train unit, and thus we refer to them as either VIRM4 or VIRM6. On the other hand, SLT trains can also be of subtype '4' or '6', which also indicates the number of carriages of the train unit and we refer to them as SLT4 or SLT6. Figure 8.1 shows the train unit types used in the experiments and some of their details. Besides, train compositions consist of a maximum of two train units. Taking into account that only trains of the same type can be combined, the set of possible train compositions $i \in I$ equals I = { [SLT4], [SLT6], [SLT4, SLT4], [SLT4, SLT6], [SLT6, SLT6], [VIRM4], [VIRM6], [VIRM4, VIRM4], [VIRM4, VIRM6], [VIRM6, VIRM6] }.

Finally, for simplification, we assume the events duration associated with movements, combination, splitting and set back as shown in Table 8.2.

Movement duration [s]	120
Combination duration [s]	120
Split duration [s]	120
Set back duration [s]	0

Table 8.2: Duration of events.

Train name	Train length	First introduction	
SLT4	69.36 m	2009	
SLT6	100.54 m		
VIRM4	108.56 m	1994	
VIRM6	162.06 m		

Figure 8.1: Train unit types and subtypes used in the experiments with details. Drawings from Treinposities (2021).

8.2.2. Objectives and Performance Metrics

In the TORS-MATDRL framework, as explained, agents aim at maximizing the total reward received during each episode, but many different feasible solutions exist for each instance. In order to measure the quality of the produced route plans on the basis of the components of the TBS and IRTS-based routing strategies, we use the following performance metrics to evaluate the contribution of each component. Additionally, we include some robustness and capacity consumption performance metrics. The performance metrics presented below may need some adaptations depending on the experiment or the routing strategy used. It should be noted that we only evaluate sets of solved instances.

1) Standard parking rules: We measure the average number of times that trains have correctly parked on parking tracks $\in T_p$ during each episode according to the TBS-based routing strategy and to the IRTS-based routing strategy parking rules, as well as the average number of parking events on empty parking tracks $\in T_p$.It is important to note that in the TUSP it is very difficult to define lower and upper bounds for such measures since it depends on various variables such as the random distribution of arriving trains. Besides, it should also be noted that this metric is independent of the track where such events take place. Agents eventually learn a policy which yields some consistency in the selection of parking tracks, as shown by Peer et al. (2018).

2) Combination and split rules: For a given instance it is possible to measure a lower bound of the amount of combinations and splits needed. However, the way combining and splitting are handled differs significantly between both routing strategies. (a) In the TBS-based routing strategy we can define a lower bound of the amount of combinations and splits needed to solve a particular instance and measure the actual number of combinations and splits executed, as well as the location where these were carried out. (b) In the IRTS-based routing strategy, the lower bound is directly given by the train matching given and hence there are only unique correct combinations and splits that must be carried out to solve the instance correctly. Since we are only analyzing correctly solved instances, then the amount of combinations and splits is already known in advance. It is however insightful to analyze where these operations take place in the shunting yard. The average number of combinations and splits are normalized on the lower bound calculated. Therefore, an average number of combinations equal to 1 means trains perform exactly the expected number of combinations. A value greater than 1 means trains do on average more combinations than what is expected.

3) Conflict resolution rules: We measure how often the conflict resolution rules are executed for each conflict, i.e. (a) late combination on gateway track and (b) relocation of train units. More specifically, for the first case, we measure how often are trains combined on the gateway track shortly before departure, and for the second case we measure how often does a train unit move from a parking track $\tau \in T_p$ to a relocation track $\tau \in T_r$, which is an indicator of a relocation movement being executed.

4) Movements: One of the main goals of the routing strategies is to minimize the number of movements. Therefore, the average number of movements during and instance, or alternatively the number of times the action *Move* is selected by, is a very illustrative metric to assess the performance of our implementations. Additionally, movements are classified by pairs of track subsets $\subset T$, e.g. from parking tracks $\tau \in T_p$ to relocation tracks $\tau \in T_r$, which is generally a relocation movement, or from parking tracks $\tau \in T_p$ to gateway track $\tau \in T_g$, which is an exit movement. We understand an entry movement as a movement originating on the gateway track $\tau \in T_g$ with destination to any track inside the shunting yard $\tau \in T$, and the opposite, from any track inside the shunting yard to the gateway track as an exit movement. This classification however depends on the shunting yard type, since we may have additional track sets or particular train flow patterns.

5) Robustness in time and space: Robustness in time and space of the produced shunting plans is also insightful in our study. We focus on (1) robustness against disturbances in train arrivals and (2) robustness against disturbances in service tasks, and we approach them proactively. Regarding the first one, we consider disturbances in train arrivals as delays in the scheduled arrival times. Such delays can have an effect on the predefined shunting plans since they can even change the train arrival sequences if these are long enough. In this work, such delays are implicitly taken into account within the variability in arrival times and sequences of the instances we generate. One possible measure of robustness is based on the work on robustness in critical points (RCP) by Solinen et al. (2017), which measured the time slack between actions at critical points. In this context, our objective is to maximize the buffer time between consecutive movements at certain critical infrastructure points of the shunting yard, so that the ability of the output shunting plans to deal with such disturbances is higher. Nevertheless, in TORS-MATDRL it is difficult to put this approach to RCP in practice due to the design of action triggers. As described and showed in chapter 5, agents are prompted to choose an action at each action trigger, and those are placed around each train arrival, around each time a service task is completed and around each departure. Consequently, the TORS-MATDRL sequential planning does not really give flexibility to move actions in time and thus to observe significant impacts on the RCP as a result of our implementations. As a result, we do not take into account this measure of robustness in this work. The second approach to robustness relates to the ability of handling disturbances and still being able to carry out the assigned service tasks. Disturbances generally relate to service tasks taking longer than expected, or additional tasks added to the planning, which can have an impact on the predefined shunt plans. However, removing service tasks from the planning does not have practical effects on the route plan, which will not need any adjustment.

In TORS-MATDRL (NS, 2020), the design to handle resource-limited service tasks (i.e. those that can only be carried out at service facilities), such as external cleaning, already prompts agents to spend the minimum required time at the service facility, following a sequence of events (1) arrival at service facility, (2) immediately start servicing and (3) immediately clear the facility as soon as servicing is completed. Therefore, there is limited potential to measure and ensure robustness in the context of such service tasks. Nevertheless, it is possible to measure the robustness associated with those service tasks that can be carried out on parking tracks, such as some technical inspections or internal cleaning. Therefore, robustness relates to having trains staying parked on parking tracks for as long as possible and not moving unnecessarily to other parking tracks, which relates to the previously described measure on the number of train movements. Maximizing the parking duration of trains increases the probability that the described disturbances can be handled without the need for adjustments in the shunt plans and hence it contributes positively towards a higher robustness of the shunt plans. Hence, we measure the number of parking events and the average duration of those.

Similarly, we can use the same measure for robustness against disturbances in service tasks on parking tracks to measure the duration of events of trains parked on certain critical tracks. We call critical tracks to those that are highly used or that we are interested to use as briefly as possible and to clear them as soon as possible for other trains to use them. Examples of such tracks include service facility tracks $\tau \in T_f$ or relocation tracks $\tau \in T_r$. Such tracks might lead to bottleneck issues and hence we are interested in minimizing the duration of each parking event taking place on these tracks.

Lastly, we calculate the average consumption of parking tracks, i.e. what is the maximum amount of parking tracks being used simultaneously during each solved instance, and how trains use the capacity on each of the used tracks in terms of train length and track length after all trains have arrived and parked in the shunting yard. These measures will draw insights on how the efficiently the overall parking capacity is used in different problem designs. We consider to be more efficient the situation in which less tracks are being used simultaneously and with a higher consumption of the parking length on these tracks.

Therefore, the performance metrics that we use in our experiments are summarized in Table 8.3. Additionally, we define the solvability as the percentage of solved instances out of a certain set of instances that were given to the programme.

Category	Subcategory	
Standard parking rules		Average number of correct TBS parking events
		Average number of correct IRTS parking events
		Average number of parking events on empty tracks
Combination and split rules		Average number of combinations normalized on lower bound
		Share of combinations per track set [%]
		Average number of splits normalized on lower bound
		Share of splits per track set [%]
Conflict resolution rules	Late combination on gateway track	Average number of combinations on gateway track
	Relocation	Average number of relocation movements
Movements		Total amount of movements
		Movements by class (depending on shunting yard)
Robustness against disturbances in service tasks		Average duration of parking events on parking tracks [s]
Capacity consumption		Average duration of parking events on critical tracks [s]
		Occupancy of critical tracks [%]
		Average consumption of parking tracks [%]
		Average consumption of parking capacity [%]

Table 8.3: List of key performance indicators used. Averages are defined per train unit.

8.2.3. Benchmarking

Our results with TORS-MATDRL in each experiment are benchmarked against the results obtained with the HIP programme developed at NS (section 3.2), which is currently the approach with the highest performance in solving the TUSP in terms of solvability. The running time in HIP is limited to two minutes for each instance, which is sufficient for HIP to find a solution. Both TORS-MATDRL and HIP have the same actions space and benchmarking is made with the same sets of instances.

8.2.4. Learning Parameters

In this sub section we specify the learning parameters used in TORS-MATDRL that have a significant effect on the behaviour of the network. First, the learning rate of the network is set at 0.00002. It is important to note regarding the tuning of the learning rate is that if the value is too small, then it takes longer to converge and there is a higher risk to fall into local minima. By contrast, if the value is too high, then it can help to jump out of local minima, but the algorithm does not properly explore the cost function space and it takes longer to converge. In this research we use the learning rate used in the precedent work by Barnhoorn (2020).

Second, the discount factor γ is set equal to 0.9999, which makes agents take highly into account future rewards (see section 2.2). Thirdly, each time an action is chosen by an agent, the instance advances one step. The network memory is capped at 125.000 steps (or experiences), from which 256 steps are sampled each time the network trains. Besides, the target network synchronizes with the weights of the main network once every 200 episodes.

With respect to experience replay, in order to give more importance to recent experiences, the sampling is made in such a way that the memory is divided in four sets, and experiences are sampled from these sets with probabilities of 0.15, 0.2, 0.3 and 0.35, increasing towards the set of the most recent experiences, so that those are sampled more frequently. This procedure ensures that the distribution over time remains constant over training epochs.

As described in section 6.1, an exploration phase is necessary to find random initial points in the search space and the extent to which we allow agents to explore randomly depends on the size of the set of instances and their complexity, and it should be large enough so that it explores the search space sufficiently. Nevertheless, it is not really possible to estimate the length of this phase by logic and we define it empirically. The ε decay (see section 2.2) indicates how much ε decays after each episode and, similarly as the length of the exploration phase, it is based on the training characteristics and its value is defined empirically. In conclusion, these learning parameters are experiment-specific and hence they are tailored for each experiment. The learning parameters described that are applicable for all experiments are summarized in Table 8.4.

Lastly, the training duration for a specific experiment cannot really be anticipated and will be stopped when a relative converge in the q-value during training is observed. New sets of output weights are printed each 200 episodes.

Learning rate α		0.00002
Batch size	[steps]	256
Update target rate	[episodes]	200
Discount factor γ		0.9999
Memory length	[steps]	125,000

Table 8.4: Main learning parameters for experiments.

8.2.5. Action Triggers

Besides, we need to define in our experiments what *Middle* triggers we want to add to the set of triggers since these will give agents the chances to learn certain action sequences that may be needed to solve the instances correctly. The *Arrival* and *Departure* triggers are predefined based on the input instances, and each *EndService* trigger is created as soon as one train unit starts a service task, since we do not know in advance when a service task will start and be completed. The *Middle* triggers are added around the previously mentioned triggers to allow trains to choose some actions and make some movements. Nevertheless, the amount of necessary *Middle* triggers and its ranking highly depends on the case study and on the routing strategy followed, since for instance in some cases we may need more movements around *Arrival* triggers, after *EndService* triggers or before *Departure* triggers. Therefore, we specify for each experiments what *Middle* triggers we add in order to give agents sufficient chances to take the correct actions to ultimately solve the instances given.

8.2.6. Violations

Episodes will end prematurely when certain violations happen. The most common violations are the following:

- Train cannot leave: at the time of departure, the corresponding train was not on the gateway track.
- Did not depart: at the time of departure, the corresponding train was on the gateway track but the agent did not choose to depart, or another train was blocking the exit side of the departure track.
- Composition not present: at the time of departure, the corresponding train composition had not been arranged yet.
- Arrival track reserved: entry track length was exceeded upon train arrival due to presence of trains parked on the entry track.

8.3. Experiment 1: Routing Strategies Performance Assessment with respect to Baseline Variant

8.3.1. Setup

In this section we assess the performance of the developed routing strategies by benchmarking them against a baseline variant defined by the TORS-MATDRL original settings, with our heuristic for random exploration, featuring the original reward function (Algorithm 1) and without routing strategies. For this, we use sets of instances with 6 trains of the same rolling stock type, but different subtypes, with service tasks and combination with two variants: (1) with 0% matching for the TBS-RS and (2) with 100% matching for the IRTS-RS. Therefore, we report four sets of results in total:

- (1) 0% matching in TORS-MATDRL without RS.
- (2) 0% matching in TORS-MATDRL with TBS-RS.
- (3) 100% matching in TORS-MATDRL without RS.
- (4) 100% matching in TORS-MATDRL with IRTS-RS.

The experiment involves a total of 6 train units of the same type arriving individually at the shunting yard and four departing trains, two consisting of single train units and the other two consisting of two pair of train units each. Therefore, two combinations have to be performed. Table 8.5 shows the list of arriving trains and departing trains. In total, 150 instances with such characteristics have been generated. We carry out this experiment in the case study of Heerlen (*carrousel* shunting yard without train flow pattern constraints).

	Arriving trains	**Departing trains**
1	[SLT4]	[SLT6]
2	[SLT4]	[SLT6]
3	[SLT6]	[SLT4, SLT6]
4	[SLT6]	[SLT4, SLT6]
5	[SLT6]	
6	[SLT6]	

Table 8.5: List of arriving and departing trains instances for Experiment 1 (unordered).

Table 8.6 shows the ranking of triggers for each instance. We use the same ranking of triggers for the approaches with RS and without RS.

Training	**0% matching**	**100% matching**
Rank	Triggers	Triggers
$i-1$		Middle
i	**Arrival**	**Arrival**
$i+1$	Middle	Middle
$j-2$	Middle	
$j-1$	Middle	Middle
j	**PrepDeparture**	**PrepDeparture**

Table 8.6: Middle triggers added in Experiment 1.

Table 8.7 shows the learning parameters used in Experiment 1. We use the same for the four trainings for this experiment.

ε decay		1 / 10,000
Exploration phase	[steps]	50,000
Update target rate	[episodes]	200
Discount factor γ		0.9999
Memory length	[steps]	125,000

Table 8.7: Learning parameters in Experiment 1.

8.3.2. Results

Table 8.8 shows the main computational results of the training processes. For each training process we report the loss function, q-value, solvability and total reward during training. Figure 8.2, Figure 8.3, Figure 8.4 and Figure 8.5 show the loss function, the q-value, solvability and total reward, respectively, during training with 0% matching and no routing strategy respectively. Figure 8.6, Figure 8.7, Figure 8.8 and Figure 8.9 show the loss function, the q-value, solvability and total reward, respectively, during training of the TBS-based routing strategy respectively. Figure 8.10, Figure 8.11, Figure 8.12 and Figure 8.13 show the loss function, the q-value, solvability and total reward, respectively, during training with 100% matching and no routing strategy respectively. Lastly, Figure 8.14, Figure 8.15, Figure 8.16 and Figure 8.17 show the loss function, the q-value, solvability and total reward, respectively, during training of the IRTS-based routing strategy respectively. Since the solvability of the models with no RS is very low, it is very difficult to compare using the exact same sets of instances. We use a testing set of 12 instances for the case with no matching and no RS, 23 instances in the case with TBS-RS the case with no matching, 11 instances for the case with predefined matching and no RS and finally a set with 22 instances with IRTS-RS.

Firstly, in the problem with 0% matching, we observe in the computational results in Table 8.8 how the TBS-RS significantly improves the learning process and hence it outperforms TORS-MATDRL with no RS in terms of a much higher solvability (up to 55% compared to less than 10% with no RS after about 5,000 training episodes). When looking at Figure 8.5, it is noticeable that the total reward is actually higher without RS (about 15-20) compared to TBS-RS (about 10-15) (Figure 8.9), which might seem counter intuitive since the latter is able to solve remarkably more instances. The reason for this is that agents overexploit the combination and split actions (see also Table 8.9), for which in the original reward function (Algorithm 1) only positive rewards are assigned when the unit will become (part of) any correct composition according to the departure sequence, and no negative rewards are given for undesired combination or split operations. As a result, agents collect a very high reward on average due to this behaviour. This is however not an issue in the problem with 100% matching since once agents have performed the correct split and/or combination (unique when the matching is predefined) and hence they are already in the correct composition for a departure, all subsequent redundant splitting and combination actions are filtered out.

Similarly, in the problem with predefined matching, we clearly observe in the computational results in Table 8.8 that the IRTS-RS can significantly guide agents to solve instances and hence to achieve a much higher solvability compared to TORS-MATDRL without RS (45% compared to less than 5%). The latter does not seem to converge after a longer amount of episodes (4,400 compared to 3,400) and training time (16 hours compared to 14 hours). Also, the arrival track reserved violation is quite revealing of its performance. The reason is that this violation is very common at the beginning of any training since agents arrive at the gateway track and have to learn to clear this track by moving inside the shunting yard towards parking tracks so that the next arriving trains can actually arrive at the gateway track. Therefore, this violation usually happens at the beginning of instances and hence agents will not be able to explore most of the instance. When we include our RS in the problem, agents usually learn quickly to avoid this violation until the point that it barely happens when the highest solvability was achieved (less than 2% of the episodes). By contrast, without RS, we see it still happens very often when we stopped the algorithm (about 25-45%). Besides, the composition not present violation (which takes place at the time of a -failed- departure) happens more often in the problem with IRTS-RS (about 29% of the episodes), which suggests that indeed, in general, agents are able to advance more during each instance and explore more exhaustively and reach the departure time. Lastly, Figures 8.10 to 8.17 also support our assessment of the computational results by showing that, with the IRTS-RS, the programme achieves significantly higher q-values, average total reward and number of solved instances. The average total reward plot also suggests that agents have not learn to progress significantly during each instance, which explains the low solvability achieved.

Table 8.9 presents the performance metrics obtained in TORS-MATDRL (1) with 0% matching without RS, (2) with 0% matching with the TBS-RS, (3) with 100% matching without RS and (4) with 0% matching with the IRTS-RS. The performance metrics in the models without RS show a lower quality of the produced route plans. Since in general there are no rewards for agents to perform movements, agents find difficult to learn most events involving movements such as correct parking or correct combination. Therefore, although some of the performance metrics such as the average number of parking events have some similarities when comparing with and without RS, we can assume that these occur simply by chance rather than as a result of a well-fundamented policy. Besides, agents perform a significantly higher average number of movements in both problems with no RS (15.5-17.0) compared to the same problems solved with RS (12.7-14.7), with plenty of potentially unnecessary relocations and repositioning movements. The lower number of arriving movements suggest that trains often stay at the gateway track and do not move inside the shunting yard (6 arriving trains, although average number of entries is below 6), which matches with the higher occurrence of the ar-

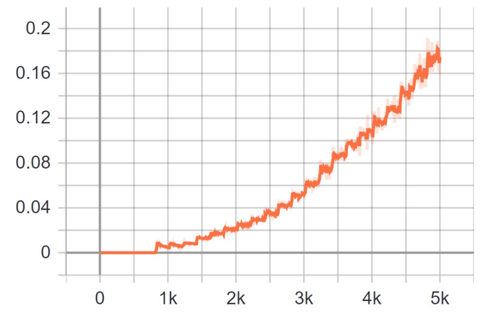

Figure 8.2: Experiment 1 0% matching no RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

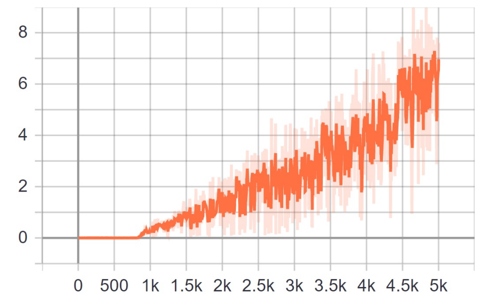

Figure 8.3: Experiment 1 0% matching no RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

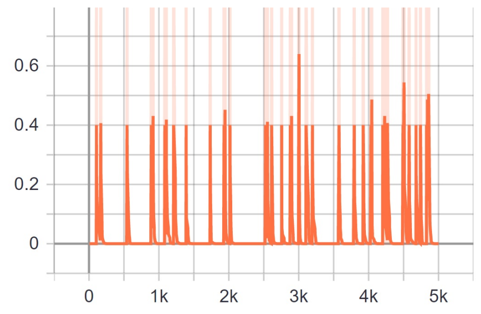

Figure 8.4: Experiment 1 0% matching no RS average solved during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

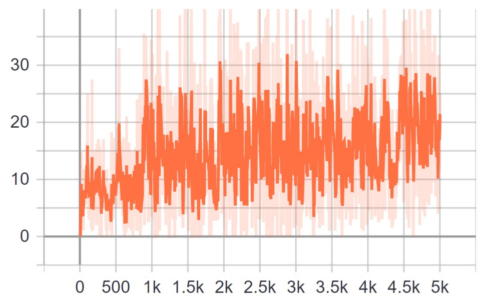

Figure 8.5: Experiment 1 0% matching no RS total reward during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

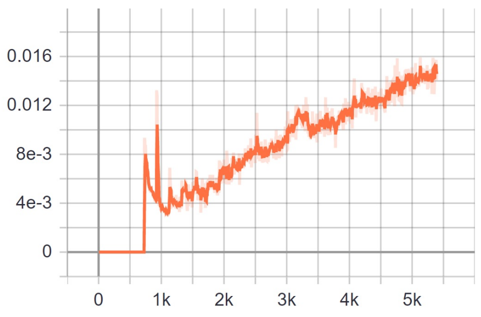

Figure 8.6: Experiment 1 0% matching TBS-RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

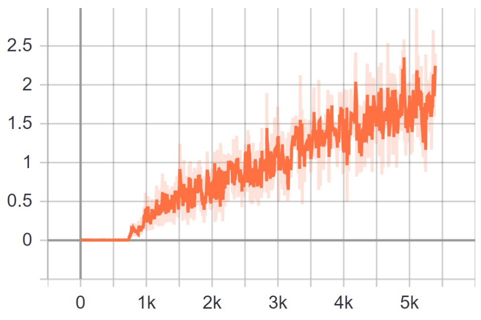

Figure 8.7: Experiment 1 0% matching TBS-RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

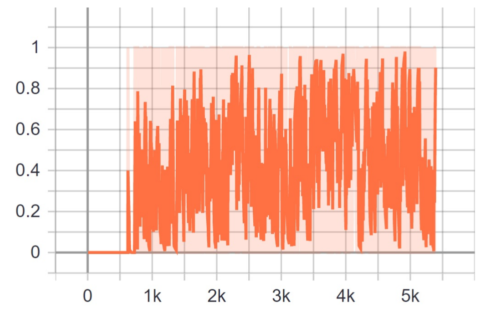

Figure 8.8: Experiment 1 0% matching TBS-RS average solved during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

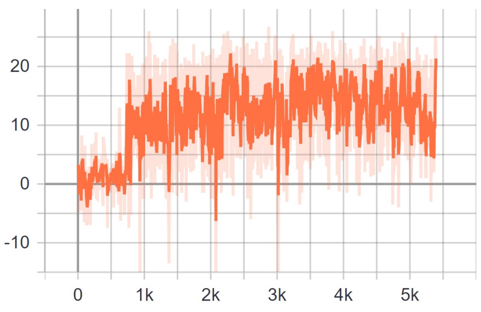

Figure 8.9: Experiment 1 0% matching TBS-RS total reward during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

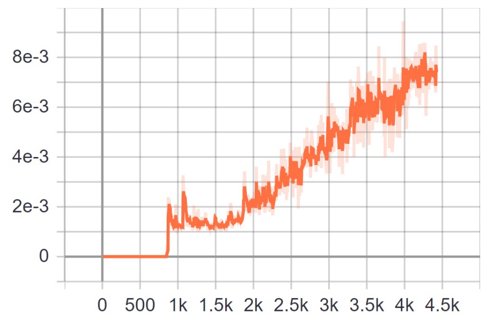

Figure 8.10: Experiment 1 100% matching no RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

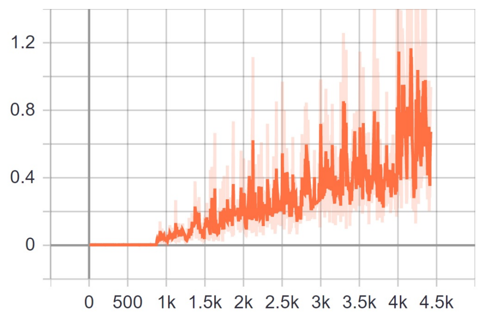

Figure 8.11: Experiment 1 100% matching no RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

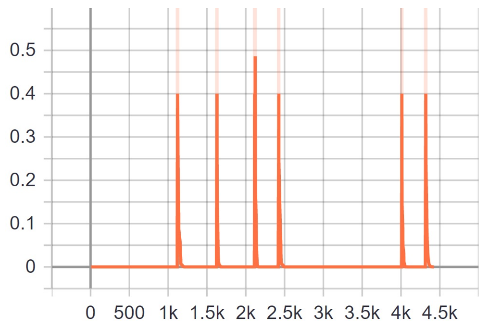

Figure 8.12: Experiment 1 100% matching no RS average solved during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

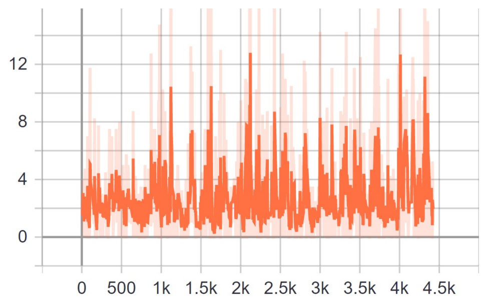

Figure 8.13: Experiment 1 100% matching no RS total reward during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

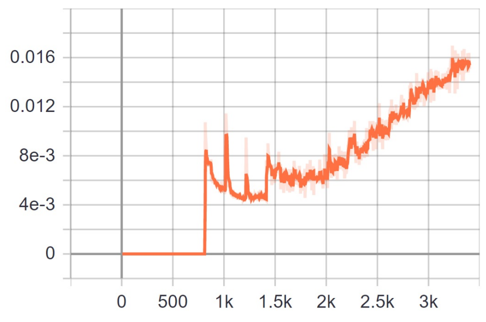

Figure 8.14: Experiment 1 100% matching IRTS-RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

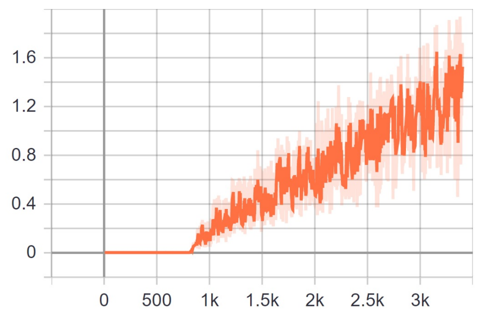

Figure 8.15: Experiment 1 100% matching IRTS-RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

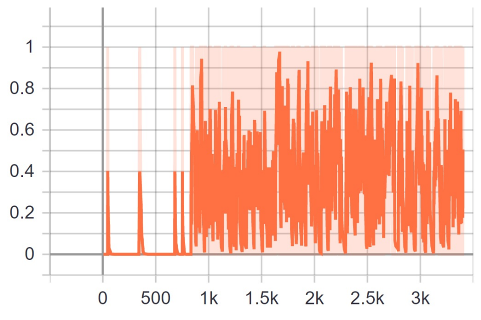

Figure 8.16: Experiment 1 100% matching IRTS-RS average solved during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

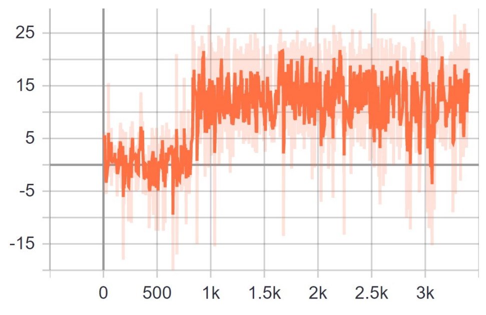

Figure 8.17: Experiment 1 100% matching IRTS-RS total reward during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

	0% matching		**100% matching**	
Training	**TORS-MATDRL No RS**	**TORS-MATDRL with TBS-RS**	**TORS-MATDRL No RS**	**TORS-MATDRL with IRTS-RS**
Number of episodes	5,000	5,200	4,400	3,400
Training time	23.0 hours	26.8 hours	16.1 hours	13.8 hours
End ε	0.6032	0.7546	0.6645	0.7924
Maximum solvability (200 episodes)	7.5%	56.0%	3.0%	44.5%
Not solved (last 200 episodes)	92.5%	44.0%	97.0%	55.5%
Violations per type:				
Train cannot leave	29.0%	24.0%	34.5%	24.0%
Did not depart	23.0%	10.5%	2.0%	2.0%
Arrival track reserved	24.0%	1.5%	46.0%	0.5%
Composition not present	14.0%	8.0%	14.0%	29.0%

Table 8.8: Computational results of the training process in Experiment 1.

rival track reserved violation (25-45% of the episodes) (Table 8.8). This behaviour has also an effect in terms of a lower consumption of parking capacity. It is important therefore to bear in mind this behaviour of trains staying on the entry track since some performance metrics could lead to misleading conclusions. Additionally, with no RS, the duration of parking events on parking tracks are shorter (7,000-8,000 seconds compared to 14,000-15,500), which show that trains keep moving around and spend more time in either the relocation track or the gateway track, which is undesired since these are critical tracks that we want to clear as soon as possible for other trains to use them for either arrivals, departures or conflict resolution. In fact, the occupancy of the relocation track (critical track in Heerlen) is significantly higher with no RS (44-52%) compared to with RS (3-13%). Lastly, the parking capacity is used on average less efficiently with no RS, i.e. more tracks used on average and/or shorter track length used.

The comparison between the results without RS for 0% matching and 100% matching also shows that the former is a more relaxed problem since the maximum solvability achieved by the agents is higher (7.5% compared to 3.0%) as well as the total reward (15-20 compared to 2-3), compared to the problem with 100% matching.

In conclusion, the results of the first experiment show that the implementation of both the TBS-RS and the IRTS-RS in TORS-MATDRL has very positive effects on the overall learning process and in the quality of the produced route plans on the basis of the defined computational performance and performance metrics, despite the models have to be trained until convergence for fair comparisons on computational performance. Both RS are able to solve up to about 45-55% of the input instances.

8.4. Experiment 2: Matching and Combination Problems with Single Rolling Stock Type in Carrousel Shunting Yard without Flow Pattern Constraints

8.4.1. Setup

In the second experiment we use the exact same sets of instances used in the first experiment and the same models obtained using the TBS-RS and the IRTS-RS. These instances involve a total of 6 train units of the same type arriving individually at the shunting yard and four departing trains, two consisting of single train units and the other two consisting of two pair of train units each. Therefore, two combinations have to be performed. Table 8.10 shows the list of arriving trains and departing trains. In total, 150 instances with such characteristics have been generated. We carry out this experiment in the case study of Heerlen (*carrousel* shunting yard without train flow pattern constraints).

The amount of *Middle* triggers needed to let agents learn certain action sequences, such as combining or resolving conflicts, depends on the routing strategy used. In both the TBS and IRTS-based routing strategy, we give one *Middle* trigger to let agents correct their composition by performing a combination right after

			0% matching		100% matching	
Category	**Subcategory**	**Performance Indicator**	**TORS-MATDRL No RS**	**TORS-MATDRL TBS-RS**	**TORS-MATDRL No RS**	**TORS-MATDRL IRTS-RS**
Standard parking rules		Average number of correct TBS parking events	3.83	5.13	3.54	3.23
		Average number of correct IRTS parking events	-	-	2.36	3.00
		Average number of empty parking events	3.00	2.69	3.45	3.32
Combination and split rules		Average number of combinations normalized on lower bound	5.38	1.05	1.00	1.00
		Combinations on parking track [%]	24.8%	91.7%	13.6%	77.0%
		Combinations on gateway track [%]	51.2%	6.3%	72.7%	23.0%
		Combinations on relocation track [%]	24.0%	2.1%	13.6%	0.0%
		Average number of splits	8.75	0.09	0.00	0.00
		Splits on parking tracks [%]	25.7%	100.0%	-	-
Conflict resolution	Late combination on gateway track	Average number of combinations on gateway track	5.50	0.13	1.45	0.45
	Relocation	Average number of relocation movements	1.58	0.95	1.09	0.32
Movements		Total amount of movements	15.50	14.74	16.80	12.68
		Average number of entry movements	5.25	6.69	5.91	6.23
		Average number of exit movements	4.33	4.82	5.36	4.68
		Average number of relocation movements (to relocation track)	1.58	0.95	1.36	0.32
		Average number of relocation movements (back to parking)	2.08	0.91	1.09	0.32
		Average number of repositioning movements	2.25	1.35	3.09	1.14
Robustness against disturbances in service tasks		Average duration of parking events on parking tracks [s]	7,765.9	13,818.8	7,046.0	15,591.4
Capacity consumption		Average duration of parking events on relocation track [s]	9,967.2	3,147.7	9,485.3	4,127.6
		Occupancy of relocation tracks [%]	52.3%	13.2%	44.4%	3.4%
		Average consumption of parking tracks [%]	58.3%	70.5%	84.1%	79.5%
		Average consumption of parking capacity [%]	38.9%	45.4%	36.5%	34.8%

Table 8.9: Performance metrics for Experiment 1.

	Arriving trains	**Departing trains**
1	[SLT4]	[SLT6]
2	[SLT4]	[SLT6]
3	[SLT6]	[SLT4, SLT6]
4	[SLT6]	[SLT4, SLT6]
5	[SLT6]	
6	[SLT6]	

Table 8.10: List of arriving and departing trains instances for Experiment 2 (unordered).

arrival. Besides, in the TBS-based routing strategy, conflicts are solved right before each *Departure* trigger. In order to let agents perform a complete relocation movement or late combination on the gateway track before departing, we give two *Middle* triggers after each *Departure* trigger. In the IRTS-based routing strategy, by contrast, conflict resolution is triggered during arrivals and take place as soon as the standard parking rules are violated. Therefore, we add one additional *Middle* trigger in between each *Arrival* trigger to let agents resolve conflicts as soon as possible. We keep one *Middle* trigger before each departure which can still allow to perform late combination on the gateway track. Table 8.11 shows the ranking of triggers for each instance. All the rankings shown apply for all *Arrival* and *PrepDeparture* triggers in the instance for both routing strategies (from now on: RS).

Table 8.12 shows the learning parameters used in Experiment 2.

8.4.2. Results

Table 8.13 show the main computational results of the training process and Figure 8.18 and Figure 8.19 show the loss function and the q-value during training of the TBS-based routing strategy respectively, and Fig-

Training	**TBS-RS 0% matching**	**IRTS-RS 100% matching**
Rank	Triggers	Triggers
$i-1$		Middle
i	**Arrival**	**Arrival**
$i+1$	Middle	Middle
$j-2$	Middle	
$j-1$	Middle	Middle
j	**PrepDeparture**	**PrepDeparture**

Table 8.11: Middle triggers added in TORS-MATDRL for Experiment 2.

ε decay		1 / 10,000
Exploration phase	[steps]	50,000
Update target rate	[episodes]	200
Discount factor γ		0.9999
Memory length	[steps]	125,000

Table 8.12: Learning parameters in Experiment 2.

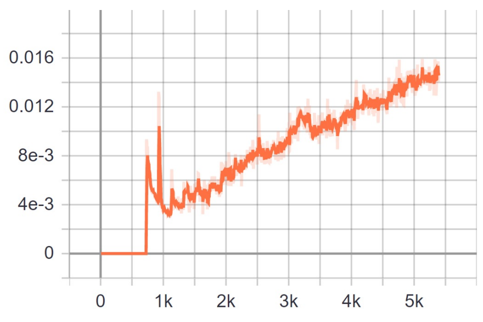

Figure 8.18: Experiment 2 TBS-RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

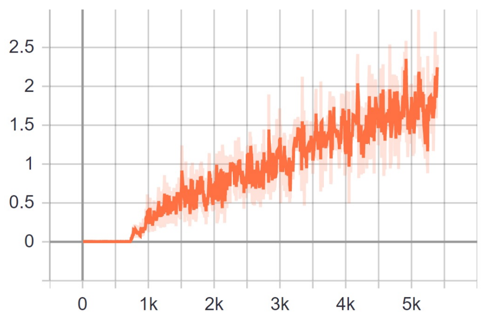

Figure 8.19: Experiment 2 TBS-RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

ure 8.20 and Figure 8.21 for the IRTS-based routing strategy.

Training	**TBS-RS 0% matching**	**IRTS-RS 100% matching**
Number of episodes	5,200	3,400
Training time	26.8 hours	13.8 hours
End ε	0.7546	0.7924
Maximum solvability (200 episodes)	56.0%	44.5%
Not solved (last 200 episodes)	44.0%	55.5%
Violations per type:		
Train cannot leave	24.0%	24.0%
Did not depart	10.5%	2.0%
Arrival track reserved	1.5%	0.5%
Composition not present	8.0%	29.0%

Table 8.13: Computational results of the training process in Experiment 2.

After testing the models obtained in TORS-MATDRL, Table 8.14 presents the performance metrics obtained in (1) TORS-MATDRL with 0% matching with the TBS-based routing strategy, (2) HIP with 0% matching, (3) TORS-MATDRL with 100% matching with the IRTS-based routing strategy and (4) HIP with 100% matching. We use a testing set of 23 instances for the case with no matching and a different testing set with 22 instances

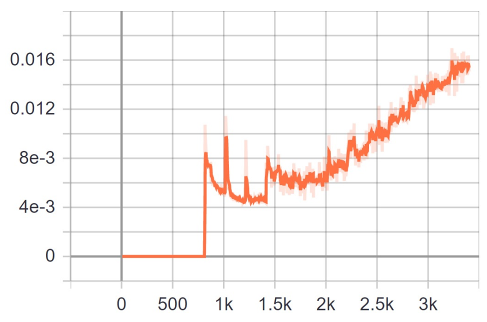

Figure 8.20: Experiment 2 IRTS-RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

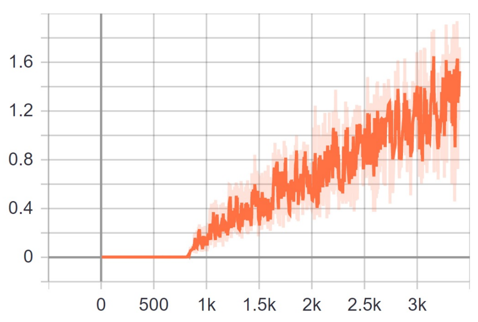

Figure 8.21: Experiment 2 IRTS-RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

for the case with predefined matching.

			0% matching		100% matching	
Category	**Subcategory**	**Performance Indicator**	**TORS-MATDRL TBS-RS**	**HIP**	**TORS-MATDRL IRTS-RS**	**HIP**
Standard parking rules		Average number of correct TBS parking events	5.13	1.96	3.23	1.91
		Average number of correct IRTS parking events	-	-	3.00	1.68
		Average number of empty parking events	2.69	2.52	3.32	2.50
Combination and split rules		Average number of combinations normalized on lower bound	1.05	1.00	1.00	1.00
		Combinations on parking track [%]	91.7%	67.4%	77.0%	70.5%
		Combinations on gateway track [%]	6.3%	21.7%	23.0%	15.9%
		Combinations on relocation track [%]	2.1%	10.9%	0.0%	13.6%
		Average number of splits	0.09	0.00	0.00	0.00
		Splits on parking tracks [%]	100.0%	-	-	-
Conflict resolution	Late combination on gateway track	Average number of combinations on gateway track	0.13	0.43	0.45	0.32
	Relocation	Average number of relocation movements	0.95	0.35	0.32	0.32
Movements		Total amount of movements	14.74	8.91	12.68	9.95
		Average number of entry movements	6.69	4.70	6.23	5.14
		Average number of exit movements	4.82	3.13	4.68	3.45
		Average number of relocation movements (to relocation track)	0.95	0.35	0.32	0.32
		Average number of relocation movements (back to parking)	0.91	0.74	0.32	0.91
		Average number of repositioning movements	1.35	0.00	1.14	0.14
Robustness against disturbances in service tasks		Average duration of parking events on parking tracks [s]	13,818.8	20,310.4	15,591.4	17,134.9
Capacity consumption		Average duration of parking events on relocation track [s]	3,147.74	6,644.0	4,127.6	7,097.8
		Occupancy of relocation tracks [%]	13.2%	22.2%	3.4%	37.2%
		Average consumption of parking tracks [%]	70.5%	63.0%	79.5%	62.5%
		Average consumption of parking capacity [%]	45.4%	29.3%	34.8%	23.1%

Table 8.14: Performance metrics for Experiment 2.

When comparing the TBS-RS with the IRTS-RS, we observe a solvability of up to 56.0% after about 27 hours of training in the former and up to 44.5% after about 14 hours of training. The training process shows a slower convergence for the TBS-RS compared to the IRTS-RS, as a result of the former being a more relaxed problem in which agents have to explore the state-action space more extensively to solve the matching problem. The increased number of possibilities to solve the problem also explains a higher solvability achieved with 0% matching. With 100% matching, by contrast, the problem is more constrained and hence the algorithm seems to converge faster.

We also observe that agents succeeded in learning to prevent blocking the entry track in both routing strategies. This is indicated by the violation "did not depart" during the sequence of departure and by "arrival track reserved" during the sequence of arrivals (in less than 2% of the episodes). Nevertheless, "did not depart" still happens in around 10% of the episodes in the TBS-RS. The violations also show that agents found less difficulty in solving the combination problem in the TBS-based routing strategy compared with the IRTS-RS. This finding supports the basic assumption behind the TBS-RS routing strategy, which is that parking trains of the same type together should help to solve the combination problem.

The most common violations are however associated with a train not being able to leave (in about 24% of the episodes), which means that at the time of its scheduled departure, the corresponding train was not on the departure track. This usually happens when there is a conflict caused by a train blocking the next train to depart (otherwise, with no conflict, the departing train would generally move correctly to the gateway track). In both the TBS-RS and the IRTS-RS, conflict resolution strategies guide agents to perform the required sequence of actions to solve conflicts when certain conditions are met. Nonetheless, it is often the case that some of these actions are delayed in time to subsequent action triggers and sometimes too late. This fact can be observed in the average duration of the parking events on relocation tracks, which is significantly long (see in Table 8.14). For the two movements associated to a relocation movement, it would make more sense to perform both movements successively with a minimal stay at the relocation track. The reason for the actual behaviour of agents is that despite of the rewards assigned, the policy learned with training does not match perfectly the expected behaviour, and other agents with higher expected q-values for certain other actions may act instead in a given action trigger.

With respect to the performance metrics (Table 8.14), there are several noticeable results. Firstly, with regards to the standard parking rules metrics, in the TBS-RS we observe that trains park correctly according to the TBS parking rules with 5.13 events on average each episode, which seems reasonable taking into account that there are 6 train units arriving at the shunting yard. Such parking events happen much less often in the IRTS-RS, which makes sense since in the IRTS-RS train are parked according to its position in the departure sequence and independently from the rolling stock type. Therefore, the amount of TBS parking events in the IRTS-RS simply happen by chance. We do not report the average number of correct IRTS parking events for the problem with no matching since at the moment train parking is taking place, the exact train units that will be in the composition of each departure are still uncertain and hence the actual departure sequence will only be known once the instance is fully solved. Therefore, this information cannot be used in advance to park trains. It should be noted that, for each RS, the number of correct parking events according to the corresponding standard parking rule is significantly higher in the TBS-RS, with 5.13 events on average, compared to the IRTS-RS, with 3.23 events on average. This fact makes sense assuming that the conditions under which correct parking is performed are more restrictive in the IRTS-RS than in the TBS-RS. In fact, the chances of a train arriving at the shunting yard and being able to park in the front of a train unit of the same rolling stock type are often higher than being able to park in the front of a train unit which is scheduled to depart later. Also, this suggests that in the IRTS-RS, arriving trains are more likely to park on an empty track. We will analyze the parking capacity consumption metrics to verify this.

Secondly, with regards to the combination and split rules, agents perform the expected number of combinations in each instance in the IRTS-RS, whereas in the TBS-RS agents succeed in performing a number of combinations very close to the lower bound, with 1.05 combinations. The reason not to perform the optimal number of combinations are the uncertainties during the problem instance associated with train departures. As a result, agents have to correct its current composition by performing a split in a very few cases agents. Combinations are generally carried out on parking tracks (80-90% of the cases), as both routing strategies prompt to do in normal conditions. For conflict resolution we allow to combine on the gateway track, which agents sometimes do to a limited extent. This conflict resolution strategy is used more often in the IRTS-RS (about 23% of the combinations), compared to the TBS-RS (about 6% of the combinations). This seems logic since in the former it is more likely to find the train to combine with, of the same rolling stock type, on a different track.

It is also interesting to note that the average total number of movements is lower in the IRTS-RS (12.7 compared to 14.7), and the main reason is a significantly lower number of relocation movements compared to the TBS-RS (about 0.3 compared to 0.9). In general, for conflict resolution, combination on the gateway track is slightly more common than relocation movements in the IRTS-RS (about 0.45 per episode compared to 0.32).

The reason for a higher number of relocation movements in the TBS-RS (about 0.9) is probably also related to the uncertainties associated with train departures, which often require relocation and repositioning movements for the correct train composition to depart. As explained previously, with 0% matching it is difficult to anticipate train departures to prevent these conflicts.

Another aspect that it is very interesting to analyze by comparing the problems with no matching and with predefined matching is how is the overall parking capacity used, in terms of number of tracks used and how is the length of each used. In this experiment we have 2 SLT4 trains and 4 SLT6 trains, which have a total train length of 544 meters. In Heerlen there are four parking tracks with a total length of 2,211 meters, which should be more than enough to solve this problem. It should be noted however that, as described in section 3.2, the actual operating capacity is reduced with the presence of service tasks and the need for combinations and splitting. The results using TORS-MATDRL show that on average, less tracks are used simultaneously in the TBS-RS compared to the IRTS-RS, with 70.5% and 79.5% of the parking tracks respectively. Additionally, the parking tracks used are used at 45.4% of its capacity with the TBS-RS and at 34.8% of its capacity with the IRTS-RS. These results mean that in general, in the TBS-RS, less parking tracks are used on average and their respective lengths are used to a higher extent, and hence the overall parking capacity is used more efficiently, in comparison to the IRTS-RS, which uses more tracks on average, with lower track length used, to handle the same amount of trains and service tasks. This supports our assumption regard the standard parking rules metrics in which in the IRTS-RS trains arriving at the shunting yard are more likely to park on an empty track since it is more difficult to perform a correct parking event.

With respect to the performance of the route plans obtained using HIP, a few things have to be noted first. It is common in the solutions produced by HIP that certain train units may stay standstill at the entry track when they do not have service tasks assigned. In the meantime, other trains may arrive or depart. This is not a problem for the feasibility of the produced route plans since, as explained, in Heerlen trains arrive in passenger service from the A-side to the station platform and access the shunting yard also from the A-side, so having a train unit standing still on the B-side of the entry track does not interfere with this ongoing traffic as long as the physical length of the entry track is not exceeded, which is of 296 meters. As a result, the overall number of movements is on average lower in the results obtained in HIP compared to TORS-MATDRL (around 9-10 compared to 13-15). Also, having trains standing still on the entry track would not be allowed in real-life assignments due to the presence of non-service traffic. By contrast, in TORS-MATDRL, all trains generally move inwards the shunting yard from the entry track since the routing strategies prompt agents to do so, which is shown by the number of entry movements, which is on average above 6, which is the number of arriving trains. Furthermore, HIP route plans feature significantly more movements from the relocation track to parking tracks $\tau \in T_p$ than from parking tracks $\tau \in T_p$ to relocation tracks (0.75-0.9 compared to 0.30-0.35). This means that the movements involving the relocation track $\tau \in T_r$ are not associated with relocation movements. Instead, trains are often directed from the entry track $\tau \in T_g$ directly to the relocation track via an empty parking track $\tau \in T_p$, where they will reverse direction and move to another parking track $\tau \in T_p$, as opposed to moving directly from the entry track to a parking track $\tau \in T_p$, which would involve less movements, avoid a saw movement and hence be more efficient since it would be less time consuming and an empty track would not be needed. Also, on average, about 10% of the combinations are performed on the relocation track.

As a result of the described behaviour of HIP regarding trains standing still on the gateway track, we also observe that the parking capacity is used to a lower extent in HIP compared to the performance with TORS-MATDRL (less than 65% of the parking tracks used on average, compared to 70-80% with TORS-MATDRL), but it is hence difficult to retrieve conclusions on how efficiently is the parking capacity used in the different approaches. Another characteristic feature of the route plans produced by TORS-MATDRL is the presence of repositioning movements. Agents are prompted to clear the facility tracks as soon as service is completed so that other trains can use them. By contrast, in HIP, trains do not clear these tracks once service is completed. As a result, the average duration of parking events on parking tracks is higher in the route plans produced by HIP (about 17,000-20,000 seconds) due to the lack of repositioning movements after servicing, compared to TORS-MATDRL (about 14,000-15,5000 seconds), which has about 1.1-1.4 of such movements on average. Note that the repair service tasks take 120 minutes (i.e. 7,200 seconds) independently of the train unit type and subtype. Lastly, it is however interesting to note that the occupancy of the relocation track is much higher in the route plans produced by HIP (22-37%) compared to TORS-MATDRL (3-13%), in which we try to minimize as much as possible.

Figure 8.22 visualizes the solution produced for one particular instance by TORS-MATDRL with the IRTS-RS. The figure represents the task schedules for each train unit in the problem, indicated by their corresponding ID on the vertical axis, and ordered from top to bottom according to their arrival times. Time is represented on the horizontal axis. Note that all trains enter and exit the shunting yard using the same track. For the sake of readability, we only label those time intervals in which the train unit stands still, i.e. during servicing and waiting. Also, any wait happening immediately after servicing, without the train moving in between, is not labelled since the location remains the same. In this particular instance, train units 2601 and 2402 combine to form one of the departing trains, as well as 2602 and 2401. Therefore, all train units within the same train perform the same actions at the same times. As an example, the detailed route of train unit 2603 of the same instance is shown in Figure 8.23. The train unit moves from platform 2 at track 205 to track 33, where it starts its scheduled service task. Once servicing is completed, it repositions on the same track in the part without service platform and reverses its direction, where it waits until shortly before departure, by moving back to track 205.

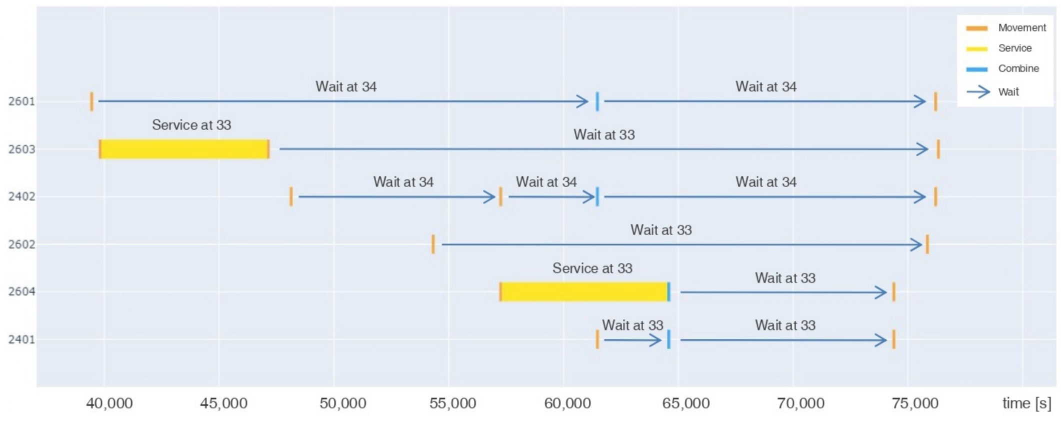

Figure 8.22: Task schedule representing the solution to one particular instance in Heerlen by TORS-MATDRL with the IRTS-RS.

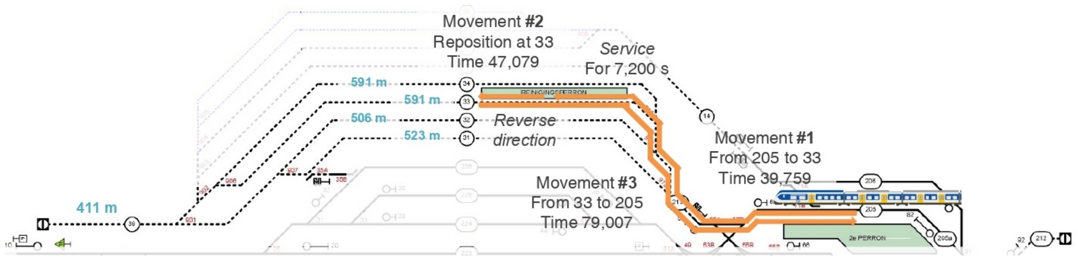

Figure 8.23: Example of the detailed route of train unit 2603 in one particular instance in Heerlen by TORS-MATDRL with the IRTS-RS. Adapted from Sporenplan Online (2019).

In conclusion, the second experiment shows the potential of the TBS-RS and the IRTS-RS to solve matching, combination and servicing problems with single rolling stock type in a *carrousel* shunting yards without flow pattern constraints. We have also shown that the problem with 100% matching is more constrained than the problem with 0% matching and hence it converges significantly faster, but it manages to solve less instances (around 10 percentage points less) and uses the overall parking capacity less efficiently. Regarding the benchmark of the performance of TORS-MATDRL against HIP, it is difficult to retrieve solid conclusions on performance due to fundamental differences on how train movements are handled in both frameworks. Nevertheless, some things can be noted. In the problem with 0% matching, HIP succeeds in meeting the exact lower bound of the number of combinations required while TORS-MATDRL does not, albeit it gets very close to it (1.05 compared to the lower bound of 1.00). Also, TORS-MATDRL often performs repositioning movements, whereas HIP generally does not. Nonetheless, TORS-MATDRL seems to perform significantly better in terms of occupying critical tracks such as the relocation track as little as possible compared to HIP.

8.5. Experiment 3: Matching and Combination Problems with Multiple Rolling Stock Types in Carrousel Shunting Yard without Flow Pattern Constraints

8.5.1. Setup

The instances used in this experiment consist of 7 train units of different types and subtypes arriving individually at the shunting yard and 6 train departures, which means that one combination has to be solved on each instance. Compared to the settings in Experiment 2, we have added one more train unit and one more rolling stock type (and hence a total of two types). Table 8.15 shows the list of arriving trains and departing trains. In total, 150 instances with such characteristics have been generated. Again we carry out this experiment in the case study of Heerlen.

	Arriving trains	**Departing trains**
1	[SLT6]	[SLT6]
2	[SLT6]	[SLT6]
3	[SLT6]	[SLT6]
4	[VIRM4]	[VIRM4]
5	[VIRM4]	[VIRM4, VIRM6]
6	[VIRM6]	[VIRM6]
7	[VIRM6]	

Table 8.15: List of arriving and departing trains instances for Experiment 3 (unordered).

Table 8.16 shows the ranking of triggers for each instance. We use the same rankings as in Experiment 2 for each routing strategy since the problem design remains the same despite of the higher number of trains and multiple rolling stock types.

Training	**TBS-RS 0% matching**	**IRTS-RS 100% matching**
Rank	Trigger	Trigger
$i-1$		Middle
i	**Arrival**	**Arrival**
$i+1$	Middle	Middle
$j-2$	Middle	Middle
$j-1$	Middle	Middle
j	**PrepDeparture**	**PrepDeparture**

Table 8.16: Middle triggers added in TORS-MATDRL for Experiment 3.

Table 8.17 shows the learning parameters used in Experiment 3.

8.5.2. Results

Table 8.18 show the main computational results of the training process and Figure 8.24 and Figure 8.25 show the loss function and the q-value, respectively, during training of the TBS-based routing strategy respectively,

ε decay		1 / 15,000
Exploration phase	[steps]	50,000
Update target rate	[episodes]	200
Discount factor γ		0.9999
Memory length	[steps]	125,000

Table 8.17: Learning parameters in Experiment 3.

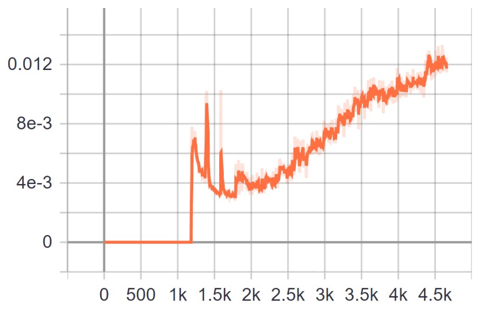

Figure 8.24: Experiment 3 TBS-RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

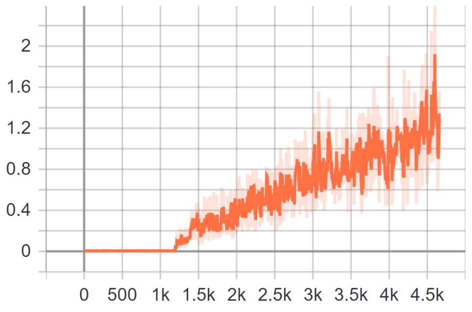

Figure 8.25: Experiment 3 TBS-RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

and Figure 8.26 and Figure 8.27 for the IRTS-based routing strategy. The performance in terms of solvability has significantly decreased compared to the computational results in Experiment 2 down to 32% after adding one more train unit and multiple rolling stock types. The main reason for this is the increased number of trains, which constraints more train movements, and the addition of multiple rolling stock types. The effect of the latter is to add additional sources of conflicts. The reason for this is that despite trains of different rolling stock types are directed to different parking tracks, when arriving trains of different rolling stock types have service tasks assigned, then these can meet on the same track where they are being serviced since both RS do not make a distinction between the available tracks for servicing. As a result, although the relocation conflict resolution strategy is applicable to such cases, the environment is anyways more constrained. Additional consequences of having a more constrained environment is that agents find more difficult to solve conflicts in both RS, which are represented by the train cannot leave violation, which goes up to about 31%. In the TBS-RS, agents found more difficult to perform correct combinations, but in the IRTS-RS, by contrast, agents have remarkably solved significantly more combination problems, with a decrease of failed combinations from 29.0% to 15.5%, for which it is more difficult to find a logic explanation. Lastly, the problem with IRTS-RS still seems to converge faster than the problem with TBS-RS, as remarked in the second experiment (section 8.4), as a similar solvability has been achieved in both RS in shorter training time with the IRTS-RS compared to the TBS-RS.

Training	**TBS-RS** **0% matching**	**IRTS-RS** **100% matching**
Number of episodes	4,600	4,400
Training time [h]	34.3 hours	40.9 hours
End ε	0.9037	0.9697
Maximum solvability (200 episodes)	32.5%	31.5%
Not solved (200 episodes)	67.5%	68.5%
Violations per type:		
Train cannot leave	31.5%	30.5%
Did not depart	19.0%	4.5%
Arrival track reserved	1.5%	18.0%
Composition not present	15.5%	15.5%

Table 8.18: Computational results of the training process in Experiment 3.

Table 8.19 presents the performance metrics obtained in (1) TORS-MATDRL with 0% matching with the TBS-based routing strategy, (2) HIP with 0% matching, (3) TORS-MATDRL with 100% matching with the IRTS-based routing strategy and (4) HIP with 100% matching. We use a testing set of 12 instances for the case with no matching and a different testing set with 10 instances for the case with predefined matching.

Firstly, looking at the results with TORS-MATDRL, the number of correct IRTS parking events in the IRTS-RS has increased from 3.0 to 4.2 due to the presence of more trains, whereas the number of correct TBS parking events in the TBS-RS has decreased from 5.1 to 4.0, probably due to the presence of multiple rolling stock

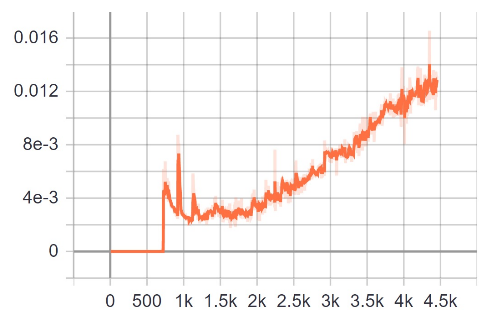

Figure 8.26: Experiment 3 IRTS-RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

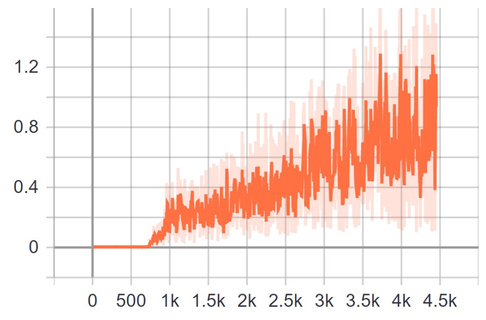

Figure 8.27: Experiment 3 IRTS-RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

types and a different mix of trains. The amount of TBS parking events with the IRTS-RS, where trains park independently from the rolling stock type, stays roughly the same (3.1 compared to 3.2 in Experiment 2). Therefore, this value simply gives an indication of the probability of parking next to a train unit of the same rolling stock type.

Secondly, with regards to combination and split rules, agents again perform the expected number of combination in each instance in the IRTS-RS, whereas in the TBS-RS agents succeed in performing a number of combinations very close to the lower bound (1.17). Combinations are generally carried out on parking tracks as expected (about 80-90% of the cases), except of the few cases in which combination is performed on the gateway track as a strategy for conflict resolution.

With regards to conflict resolution strategies, it is noticeable that combination on gateway track is scarcely performed (0.0-0.1 times on average per episode). One possible explanation for this is that with the presence of multiple rolling stock types, the chances of finding the right candidate train units for the next departure at the front of different queues of trains are lower. Besides, relocation is performed significantly more often in IRTS-RS than in TBS-RS (1.2 times on average per episode compared to 0.4), while in Experiment 2 it was pretty much the opposite. It is difficult to find a logic explanation for that.

In terms of movements, the observed behaviour is also quite similar to the behaviour observed in Experiment 2, with roughly the expected number of entry and exit movements according to the list of arriving and departing trains (i.e. 7 entries, 6 exits) and some repositioning movements (1.5-2.5). It is however remarkable that the IRTS-RS has a higher number of movements on average (18.2) compared to the TBS-RS (16.5), by contrast to what we observed in Experiment 2. The main reason for this is a significantly higher number of relocation movements in the problem with IRTS-RS and lower number of relocation movements with the TBS-RS, compared to Experiment 2.

Lastly, with regards to the parking capacity consumption, we observe again that the TBS-RS uses the parking capacity on average more efficiently than the IRTS-RS, i.e. less tracks are used on average (70.8% compared to 80.0%) and their respective lengths are occupied to a higher extent (57.0% compared to 51.8%). The performance metrics also show that the occupancy of the relocation track (critical track in Heerlen) is much lower using the TBS-RS compared to the IRTS-RS. Actually, in the IRTS-RS, trains relocating find difficult to learn to clear the relocation track as soon as possible, which explains the relatively long duration of the parking events on the relocation track (about 3,500 seconds on average), which we expected to be very low. Nonetheless, the relocation rules in the TBS-RS prompt trains to relocate right before the departure, so agents really have to ensure a very short stay at the relocation track in order to depart on time, which is what we observe in the performance metrics (about 181 seconds on average).

Regarding the performance with HIP, we observe again the same behaviour as we have seen in Experiment 2 in which trains stay standstill on the entry track and hence the overall number of movements inside the shunting yard itself is lower compared to TORS-MATDRL, which generally moves all trains inside the shunting yard. As a result, it is again difficult to compare some of the aspects that characterize the quality of the produced route plans. That being said, the results suggest that HIP tends to park trains on empty tracks rather than on partially filled tracks compared to TORS-MATDRL which parks trains on partially filled tracks, according to the standard parking rules, whenever it is possible. This is supported by the average number of parking events of each type on each problem and the parking capacity consumption metrics, which shows a less efficient use of the capacity in HIP compared to TORS-MATDRL, since more tracks occupied in HIP on average (80-90% of the tracks) and with less trains each (30-35% of their length). Regarding the combination and split rules, HIP again meets exactly the lower bound of train combinations, while TORS-MATDRL gets close to it with 1.17 combinations per episode on average (the lower bound is 1.00). Consequently, some splits are also performed to compensate for this excess (0.17 splits on average). Besides, TORS-MATDRL performs most combinations on parking tracks (90-100%) whereas HIP still performs a minority of combinations on the relocation track (15-30%) and on the gateway track (0-10%). Lastly, similarly as we observed on Experiment 2, TORS-MATDRL often performs repositioning movements (1.5-2.5 on average) while HIP usually does not (0.0-0.1), and consequently the average duration of parking events is lower in TORS-MATDRL (about 16,000-17,000 seconds) compared to HIP (about 20,000 seconds). Additionally, the relocation track is occupied much less (less than 10% compared to 85-95%) and the overall parking capacity is used more efficiently in TORS-MATDRL compared to HIP.

In conclusion, in the third experiment we have shown the potential of the TBS-RS and the IRTS-RS to solve matching, combination and servicing problems with multiple rolling stock type in a *carrousel* shunting yards without flow pattern constraints. In particular, we have observed the effects of an increased number of trains in the shunting yard and of multiple rolling stock types, which constrain more the problem but both the TBS-RS and the IRTS-RS are still able to handle and solve. With the training carried out we achieved solvabilities above 30% for problems with 7 train units. When comparing with the performance in HIP, we observe similar behaviour and hence the same conclusions as in Experiment 2.

			0% matching		**100% matching**	
Category	**Subcategory**	**Performance Indicator**	**TORS-MATDRL TBS-RS**	**HIP**	**TORS-MATDRL IRTS-RS**	**HIP**
Standard parking rules		Average number of correct TBS parking events	4.0	1.4	3.1	1.3
		Average number of correct IRTS parking events	-	-	4.2	1.2
		Average number of empty parking events	2.9	3.3	3.5	3.6
Combination and split rules		Average number of combinations normalized on lower bound	1.17	1.00	1.00	1.00
		Combinations on parking track [%]	100.0%	75.0%	90.0%	70.0%
		Combinations on gateway track [%]	0.0%	8.3%	10.0%	0.0%
		Combinations on relocation track [%]	0.0%	16.7%	0.0%	30.0%
		Average number of splits	0.2	0.0	0.0	0.0
		Splits on parking tracks [%]	100.0%	-	-	-
Conflict resolution	Late combination on gateway track	Average number of combinations on gateway track	0.0	0.1	0.1	0.0
	Relocation	Average number of relocation movements	0.4	0.5	1.2	0.4
Movements		Total amount of movements	16.5	12.3	18.2	12.7
		Average number of entry movements	7.1	6.3	7.6	6.5
		Average number of exit movements	6.1	5.4	6.7	5.5
		Average number of relocation movements (to relocation track)	0.4	0.5	1.2	0.4
		Average number of relocation movements (back to parking)	0.4	0.0	1.2	0.3
		Average number of repositioning movements	2.5	0.1	1.5	0.0
Robustness against disturbances in service tasks		Average duration of parking events on parking tracks [s]	16,260.4	20,400.1	17,059.7	20,116.1
Capacity consumption		Average duration of parking events on relocation track [s]	181.0	17,328.3	3,467.3	19,882.6
		Occupancy of relocation tracks [%]	0.3%	83.8%	9.3%	93.5%
		Average consumption of parking tracks [%]	70.8%	81.3%	80.0%	90.0%
		Average consumption of parking capacity [%]	57.0%	34.4%	51.8%	32.7%

Table 8.19: Performance metrics for Experiment 3.

8.6. Experiment 4: Matching, Combination and Splitting Problems with Multiple Rolling Stock Types in Carrousel Shunting Yard with Flow Pattern Constraints

8.6.1. Setup

The instances used in this experiment consist of a total of 9 train units, with 8 trains compositions of different rolling stock types and subtypes arriving at the shunting yard and 8 train departures. Table 8.20 shows the list of arriving trains and departing trains, which creates the need to solve one combination and one splitting during each instance. In total, 150 instances with such characteristics have been generated. We carry out this experiment in the case study of Kleine Binckhorst (*carrousel* shunting yard with train flow pattern constraints).

	Arriving trains	**Departing trains**
1	[SLT4]	[SLT4]
2	[SLT6]	[SLT6]
3	[SLT6]	[SLT6]
4	[VIRM4]	[VIRM4]
5	[VIRM4]	[VIRM4, VIRM4]
6	[VIRM4, VIRM6]	[VIRM6]
7	[VIRM6]	[VIRM6]
8	[VIRM6]	[VIRM6]

Table 8.20: List of arriving and departing trains instances for Experiment 4 (unordered).

Table 8.21 shows the ranking of triggers for each instance. In both the TBS-RS and the IRTS-RS, we give agents one *Middle* trigger after each *Arrival* trigger to make sure they can clear the gateway track after arriving and to move directly to the relocation track $\tau \in T_r$ if they do not have service tasks assigned. However, most trains will have service tasks in the facility tracks $\tau \in T_s$ (11 and 12). We do not add a *Middle* trigger before each *Arrival* since arrivals in Kleine Binckhorst tend to happen very close in time and adding this trigger does not really give any advantage to agents. We add two *Middle* triggers after each *EndService* trigger so that agents have sufficient triggers (1) to clear the facility tracks $\tau \in T_s$ by moving to the relocation track $\tau \in T_r$, (2) to reverse direction on the relocation track and (3) to clear the relocation track by moving to the parking tracks $\tau \in T_p$. With this setting, agents should also have sufficient chances to perform a combination or splitting, especially taking into account that there are other trains movements taking place simultaneously and hence more triggers that other trains can use. Lastly, for the IRTS-RS we give one more *Middle* trigger before each *Departure*, which should be sufficient for any train in the shunting yard to depart correctly on time. It is important to note that adding too many triggers to instances (i.e. more than what is really necessary) may have undesired effects on the behaviour of agents such as performing unnecessary movements as well as higher computational time to solve them.

Training	**TBS-RS 0% matching**	**IRTS-RS 100% matching**
Rank	Trigger	Trigger
i	**Arrival**	**Arrival**
$i+1$	Middle	Middle
j	**EndService**	**EndService**
$j+1$	Middle	Middle
$j+2$	Middle	Middle
$k-2$	Middle	
$k-1$	Middle	Middle
k	**PrepDeparture**	**PrepDeparture**

Table 8.21: Middle triggers added in Experiment 4.

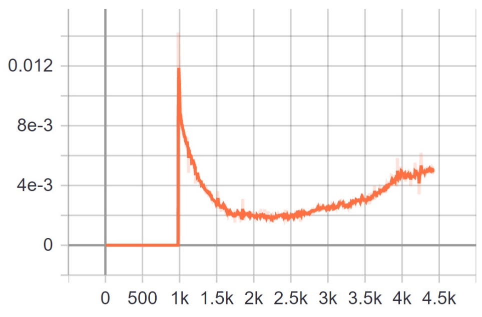

Figure 8.28: Experiment 4 TBS-RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

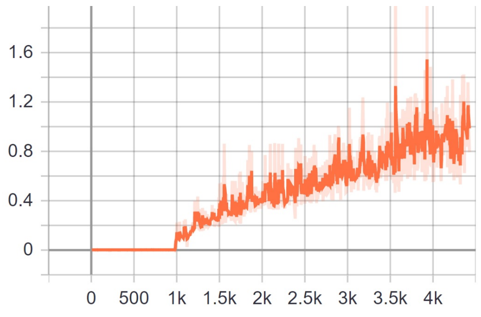

Figure 8.29: Experiment 4 TBS-RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

Table 8.22 shows the learning parameters used in Experiment 4.

Exploration phase	[steps]	80,000
Memory length	[steps]	125,000
ε decay		1 / 12,000
Discount factor γ		0.9999
Update target rate	[episodes]	200

Table 8.22: Learning parameters in Experiment 4.

8.6.2. Results

Table 8.23 shows the main computational results of the training process and Figure 8.28 and Figure 8.29 show the loss function and the q-value, respectively, during training of the TBS-based routing strategy respectively, and Figure 8.30 and Figure 8.31 for the IRTS-based routing strategy.

Training	**TBS-RS 0% matching**	**IRTS-RS 100% matching**
Number of episodes	4,400	4,800
Training time [h]	74.9 hours	69.9 hours
End ε	0.7767	0.8194
Maximum solvability (200 episodes)	15.0%	14.5%
Not solved (200 episodes)	85.0%	85.5%
Violations per type:		
Train cannot leave	42.5%	25.0%
Did not depart	12.0%	16.5%
Arrival track reserved	5.5%	19.0%
Composition not present	22.5%	23.5%

Table 8.23: Computational results of the training process in Experiment 4.

After testing the models obtained in TORS-MATDRL, Table 8.24 presents the performance metrics obtained in (1) TORS-MATDRL with 0% matching with the TBS-based routing strategy, (2) HIP with 0% matching, (3) TORS-MATDRL with 100% matching with the IRTS-based routing strategy and (4) HIP with 100% matching. We use a testing set of 14 instances for the case with no matching and a different testing set with 15 instances for the case with predefined matching. Those metrics referring to parking events, parking movements and parking tracks refer exclusively on the parking tracks $\tau \in T_p$ (i.e. tracks 52 to 55). Although it is also allowed to use the entry tracks $\tau \in T_e$ for parking, we will focus on the former set of tracks. Note that we understand as *turnaround movements* those that have the relocation track as destination from either the entry tracks $\tau \in T_e$ or the facility tracks $\tau \in T_s$, the *parking movements* as movements from the relocation tracks $\tau \in T_r$ to the parking tracks $\tau \in T_p$, the *relocation movements* as the opposite to the previous (i.e. from $\tau \in T_p$ to $\tau \in T_r$) and

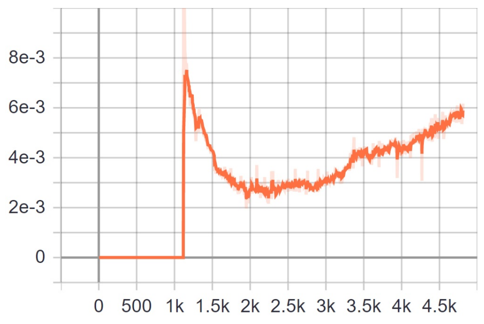

Figure 8.30: Experiment 4 IRTS-RS loss function during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

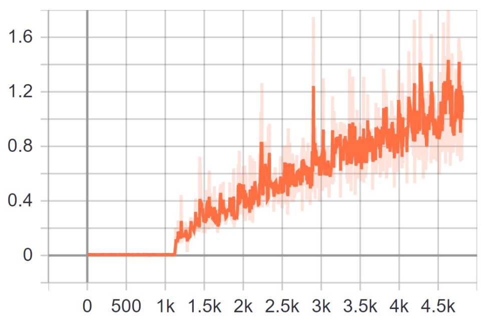

Figure 8.31: Experiment 4 IRTS-RS q-value during training. Lines generated using Tensorboard with smoothing parameter set to 0.6.

reversal movements as movements from the relocation track $\tau \in T_r$ or facility tracks $\tau \in T_s$ towards the entry tracks $\tau \in T_e$ (which go in the opposite direction as the predefined flow in this shunting yard).

In the computational results we observe lower solvabilities compared to the previous experiments, which now reach a maximum of about 15%. We also observe a relatively high amount of violations associated with trains being unable to leave (from 25 to 42% depending on the RS). The main contributing factor towards these violations is a bottleneck that happens too often on the relocation track. According to both the TBS and IRTS-RS adapted for Kleine Binckhorst, most trains have to use the relocation track to move to the parking tracks $\tau \in T_p$ (52 to 55) after arrival and servicing in a clockwise direction. The RS guide agents to clear the relocation track $\tau \in T_r$ as soon as possible every time a train arrived and reversed its direction on it. Nonetheless, it is often the case that some of these actions are delayed in time to subsequent action triggers and sometimes too late. This fact can be observed in the average duration of the parking events on relocation tracks, which is significantly long (about 1,700-2,700 seconds). For the movements associated with using the relocation track (right before and right after using it), it would make more sense to perform both movements successively with a minimal stay at the relocation track. The reason for the actual behaviour of agents is that in spite of the rewards assigned, the policy learned during training does not match perfectly the expected behaviour, and other agents with higher expected q-values for certain other actions may act instead in a given action trigger. On the positive side, agents in general learned to prevent blocking the entry track in both routing strategies. This is indicated again by the violation "did not depart" during the sequence of departure (12-16% of the cases) and by "arrival track reserved" during the sequence of arrivals (5-19% depending on the RS).

With respect to the performance metrics (Table 8.24), there are noticeable results to discuss. Firstly, with regards to the standard parking rules metrics, we observe that trains in the TBS-RS park according to the TBS parking events 5.14 times on average each episode, whereas trains in the IRTS-RS park according to the IRTS parking events 2.60 times on average. Again TBS parking events happen much more often than IRTS parking events due to the higher probability to find trains of the same rolling stock type on parking tracks (TBS) compared to the more restrictive conditions in IRTS of finding train unit scheduled to depart later than the arriving train. It is however more difficult in Kleine Binckhorst to relate this fact to a higher chance to park on an empty track since the occurrence of such parking event is very similar in both RS, with a rate close to 4. This leads us to more interesting facts. The RS prompt all trains in Kleine Binckhorst to park on the parking tracks $\tau \in T_p$ after using the relocation track $\tau \in T_r$, as long as some feasibility preconditions hold. If parking events on empty tracks happen nearly 4 times on average during each episode, and we have four parking tracks available, this suggests that in most cases, all tracks will be used since, as the first train arriving to each parking track $\tau \in T_p$ will cause such parking events.

Secondly, with regards to the combination and split rules, agents perform exactly the expected number of combinations in each instance in both the TBS and the IRTS-RS, whereas in the previous experiments the TBS-RS was performing slightly more combinations on average than the lower bound. It is difficult to explain the reason of the better performance of combinations in this case study, but it is probably influenced by the fact that in Kleine Binckhorst we do not have service facilities mixed with parking tracks as we have in Heerlen, which generally makes easier to sort trains on the parking tracks $\tau \in T_p$. Besides, in both the TBS and the IRTS-RS, combinations occur always on parking tracks $\tau \in T_p$ and splits occur always on relocation tracks $\tau \in T_r$ as it was designed in the RS.

Thirdly, with respect to the conflict resolution rules, we observe some relocation movements in the IRTS-RS (about 1.3 on average), although such movements were not implemented in that RS, which suggests that they were probably generated unnecessarily. Relocation is still allowed however in the TBS-RS and sometimes performed but not to a high extent (about 0.7). In the IRTS-RS it is much more common for trains to perform reversal movements (1.3) than in the TBS-RS (0.0). Nevertheless, the frequency of reversal movements, i.e. to move in the opposite direction to the predefined train flow in Kleine Binckhorst, is higher than trains are actually departing from tracks different from the parking tracks $\tau \in T_p$ (0.65 departures on average per episode). This suggests that again have unnecessary movements being generated.

Fourthly, regarding movements, we observe roughly the expected number of arrival and departure movements for both RS (i.e. 8 arrivals and 8 departures), as well as some departures originating on tracks different from the parking tracks $\tau \in T_p$ (about 8% of the departures), as we are allowing agents to do to relax the problem computationally and to increase the actual parking capacity. As described above, it is likely that some of the internal movements in the shunting yard are unnecessary. In fact, assuming that most trains need in principle at least four movements to close the complete *carrousel* circle ((1) entry from $\tau \in T_g$ to $\tau \in T_s$, (2) turnaround from $\tau \in T_s$ to $\tau \in T_r$, (3) parking from $\tau \in T_r$ to $\tau \in T_p$ and (4) exit from $\tau \in T_p$ to $\tau \in T_g$), then we would expect to have an average of at least 32 movements per episode (for a total of 8 trains). The results obtained show that the number of movements is indeed slightly above this expected value, with 35.4 in the TBS-RS and 39.3 in the IRTS-RS.

Fifthly, with respect to the overall parking capacity consumption, we observe very similar values for both the TBS and the IRTS-RS (about 90-95% of the parking tracks used, at 70% of its length each), whereas in the previous experiments we have seen more significant differences between RS. One plausible explanation for this is the congestion created in this experiment. In fact, in this experiment we have 9 train units in total, compared to a maximum of 7 in the case study of Heerlen. Besides, the available parking track length in Heerlen is greater than the available parking track we are using in Kleine Binckhorst with tracks 52 to 55. In particular, the total train length used in Experiment 3 was 845 meters and the parking tracks used in Heerlen had a total of 2,211 meters. In the current experiment, the total train length is 1,085 meters and the total length of the parking tracks being used in Kleine Binckhorst is 1,623 meters. Therefore, the parking tracks 52 to 55 in Kleine Binckhorst are in general more congested in the current experiment. This is likely to reduce the differences in how efficiently the parking capacity is consumed. Lastly, the critical track in Kleine Binckhorst, i.e. the relocation track, is heavily used (occupied about 37-56% of the time on average) as a result of the train flow pattern according to which most trains will use the relocation track $\tau \in T_r$. It is also remarkable that the average duration is quite long (1,700-2,700 seconds on average) despite we expected to see trains clearing it immediately after reversing the train's direction. This goes in line with the bottleneck issue of this shunting yard described previously.

Sixthly, regarding the performance with HIP, we can draw a number of interesting insights in the characteristics of the output route plans compared to those of TORS-MATDRL. HIP seems to be much more likely to use the entry tracks $\tau \in T_e$ and the facility tracks $\tau \in T_r$ to dispatch trains (about 70-75% of the departures on average), rather than the parking tracks $\tau \in T_p$ (less than 30%). As a result of this, the parking tracks $\tau \in T_p$ are less used in general and also the average number of movements is lower since trains often do not close the *carrousel* circle (about 23-24 movements in total on average). Nevertheless, combinations are still much more likely to be solved on parking tracks $\tau \in T_p$ (70-90%), for which it is difficult to find a logic explanation. Splits are also generally performed on parking tracks $\tau \in T_p$ (90-100%), by contrast to TORS-MATDRL which always performs them on the relocation track $\tau \in T_r$. With regards to movements, we observe that it is still common in HIP to perform reversal movements (2.2-2.9 on average), which makes sense since most train units need to be serviced in the instances we are using and will move back to the entry tracks $\tau \in T_e$ from there. Lastly, similarly as in the previous experiments, we observe how HIP uses more intensively the relocation track $\tau \in T_r$ (occupied about 63-75% of the time on average). This is in principle undesired, but since in the route plans produced by HIP trains will often never use the relocation track $\tau \in T_r$, then it is not that much of a concern. Also, parking durations are on average longer than those obtained with TORS-MATDRL.

In essence, the key difference in how movements are handled in HIP and TORS-MATDRL is that the latter is focused on filling up the parking tracks $\tau \in T_p$ as much as possible before using the entry tracks $\tau \in T_e$ for parking. With the amount of train units in our instances, it is generally possible to park most trains in the parking tracks $\tau \in T_p$ and hence the entry tracks $\tau \in T_e$ are used to a low extent. By contrast, HIP considers parking on entry tracks $\tau \in T_e$ much more flexibly and hence trains are parked much more spreaded in the shunting yard, as described above, rather than stacking them as much as possible as they are arriving at the shunting yard.

Figure 8.32 visualizes the solution produced for one particular instance by TORS-MATDRL with the TBS-RS. The figure represents the task schedules for each train unit in the problem, indicated by their corresponding ID on the vertical axis, and ordered from top to bottom according to their arrival times. Time is represented on the horizontal axis. Note that all trains enter and exit the shunting yard using the same track. For the sake of readability, we only label those time intervals in which the train unit stands still, i.e. during servicing and waiting. In this particular instance, train units 8602 and 9403 arrive combined and split shortly after arrival (at time 49,725), whereas units 9401 and 9402 combine to form one of the departing trains. Therefore, all train units within the same train perform the same actions at the same times. As an example, the detailed route of train unit 2401 of the same instance is shown in Figure 8.33. The train unit moves from the gateway track to track 61, where it starts its scheduled service task. Once servicing is completed, it moves to track 63 where it reverses direction and then moves to track 53 for parking. The train unit waits on track 53 until shortly before departure, by moving forward to the gateway track.

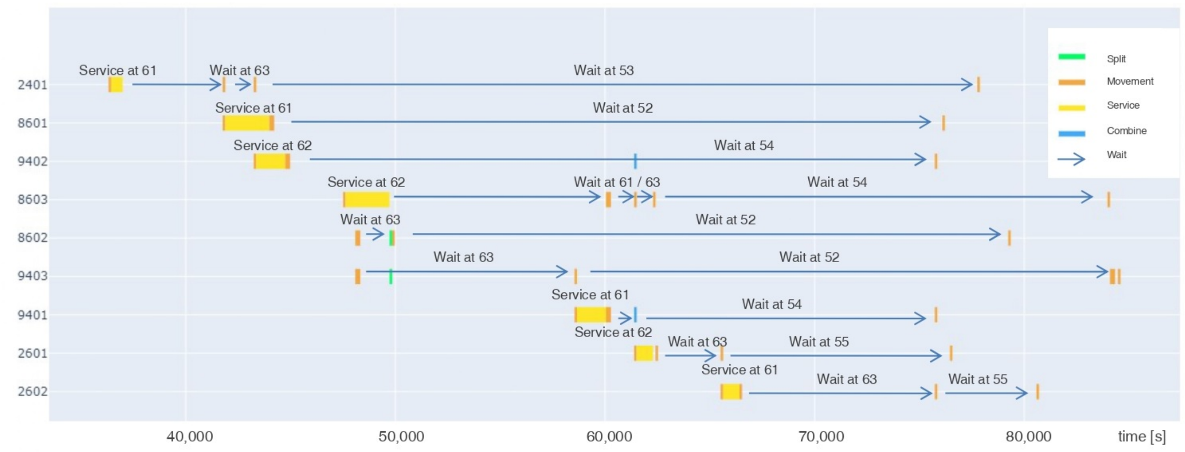

Figure 8.32: Task schedule representing the solution to one particular instance from Experiment 4 in Kleine Binckhorst.

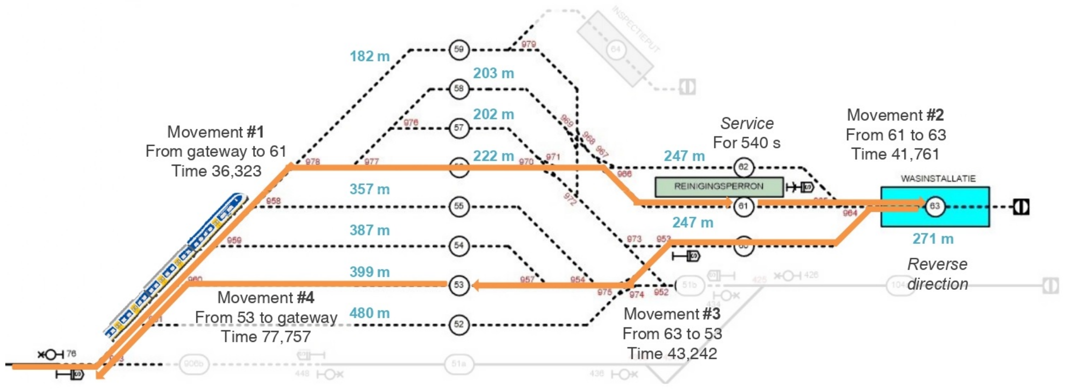

Figure 8.33: Example of the detailed route of train unit 2401 in one particular instance in Kleine Binckhorst by TORS-MATDRL with the TBS-RS. Adapted from Sporenplan Online (2019).

In conclusion, the fourth experiment has shown the potential of the TBS-RS and the IRTS-RS to solve matching, combination, splitting and servicing problems with multiple rolling stock types in a *carrousel* shunting yard with flow pattern constraints. The problem is computationally expensive since training time takes much longer than in the previous experiments to reach similar performances (we have run trainings up to 72 hours), and the relocation track creates a remarkable bottleneck that the RS find difficult to prevent. As a result of this, the RS only achieve a solvability of about 15%. On the positive side, leaving the bottleneck aside, agents generally follow the TBS and IRTS-RS rules and hence behave as expected. Also, the parking capacity is more congested in this experiment and it is used to a similar extent with both the TBS and IRTS-RS, which suggests that less differences will be observed in the parking capacity consumption between both RS as the congestion increases. Nonetheless, it is important to note that despite the congestion of the parking tracks, the entry tracks give room for additional parking capacity with 809 additional meters of track length and hence the problem is scalable to a higher amount of train units.

At last, we observe fundamental differences between TORS-MATDRL and HIP on how movements are handled, where the former tries to start filling in the parking tracks whereas HIP fills in both the entry tracks and the parking tracks. As a result, HIP spreads trains in the shunting yard more than TORS-MATDRL, which tries to stack them as much as possible, but the average number of movements is significantly lower in HIP.

			0% matching		**100% matching**	
Category	**Subcategory**	**Performance Indicator**	**TORS-MATDRL TBS-RS**	**HIP**	**TORS-MATDRL IRTS-RS**	**HIP**
Standard parking rules		Average number of correct TBS parking events	5.14	1.79	3.33	1.73
		Average number of correct IRTS parking events	-	-	2.60	1.53
		Average number of empty parking events	3.93	1.00	3.80	0.66
Combination and split rules		Average number of combinations normalized on lower bound	1.0	1.0	1.0	1.0
		Combinations on parking track [%]	100.0%	92.9%	100.0%	73.3%
		Combinations on entry track [%]	0.0%	7.1%	0.0%	26.6%
		Combinations on relocation track [%]	0.0%	0.0%	0.0%	0.0%
		Average number of splits normalized on lower bound	1.0	1.0	1.0	1.0
		Splits on parking tracks [%]	0.0%	93.3%	0.0%	100.0%
		Splits on relocation tracks [%]	100.0%	6.6%	100.0%	0.0%
Conflict resolution	Relocation	Average number of relocation movements	0.7	0.4	0.7	0.2
	Reversal	Average number of reversal movements	0.0	2.2	2.8	2.9
Movements		Total amount of movements	35.4	23.4	39.3	24.1
		Average number of entry movements	8.3	8.8	8.9	9.1
		Average number of exit movements	8.3	8.8	8.9	9.1
		Exits from parking tracks [%]	92.3%	27.6%	91.9%	27.7%
		Exits from entry tracks [%]	0.9%	18.7%	2.4%	22.6%
		Exits from rest of tracks [%]	6.8%	53.7%	5.7%	49.7%
		Average number of turnaround movements	8.6	2.1	9.3	2.1
		Average number of parking movements	9.4	1.1	9.5	0.6
		Average number of relocation movements (to relocation track)	0.7	0.4	1.3	0.2
		Average number of reversal movements	0.0	2.2	1.3	2.9
Robustness against disturbances in service tasks		Average duration of parking events on parking tracks [s]	19,331.0	23,234.7	18,260.6	19,621.6
Capacity consumption		Average duration of parking events on relocation track [s]	2,688.2	9,772.3	1,690.2	11,916.4
		Occupancy of relocation tracks [%]	55.7%	62.6%	36.8%	75.2%
		Average consumption of parking tracks [%]	94.6%	46.4%	91.6%	38.3%
		Average consumption of parking capacity [%]	70.5%	71.1%	70.9%	74.4%

Table 8.24: Performance metrics for Experiment 4.

8.7. Discussion

In this section we further discuss several of the relevant findings and observations of the experiments carried out in the previous sections.

Firstly, while matching and combination are essential aspects of the TBS-based Routing Strategy, these are not that relevant in IRTS due to the following reasons. Since there is predefined matching then there is no matching subproblem to solve, then all train units know in advance what departing train they have to be part of. Consequently, if there is any train combination required, the train units involved in the combination also know in advance the exact train unit ID next to which they have to park and combine with. Therefore, combination is simpler and the combination subproblem can be solved as soon as the involved train units have arrived at the shunting yard.

Secondly, the computational results in Experiment 2 (section 8.4) highlight the fact that in the problem with no predefined matching, the solvability is significantly higher than in the problem with predefined matching since the former is a more relaxed problem in which agents can explore many different options to solve combination problems, which also explains the longer time for the algorithm to achieve comparable performances. By contrast, with predefined matching, agents only have one possible option to solve each combination problem and hence the problem is more constrained and the algorithm converges faster. Besides, when the problem complexity increases, in terms of the amount of train units handled in the shunting yard, the problem becomes more constrained and this is reflected by the lower solvability achieved by both RS. Additionally, problems with multiple rolling stock types generally add some additional sources of conflicts. It is however necessary to run the algorithms until convergence to make fair comparisons and to retrieve solid conclusions in computational performance.

Thirdly, the formulation of the RS in TORS-MATDRL revealed that it is sometimes difficult to guide agents to perform certain sequences of actions, especially for conflict resolution strategies such as relocation. Although careful work has been carried out in terms of reward shaping, the experience has shown it is sometimes difficult to have complete control on the q-values that optimal policy learns in the long term and other agents with higher expected q-values for certain other actions may act instead in a given action trigger. As a result, the actual behaviour of agents is not always what we expected on the basis of the rewards given to them. This behaviour is also partly influenced by the action triggers design, which in certain cases may not be optimal.

Fourthly, each problem design in our experiments uses one particular routing strategy, but these are not necessarily the only possible option. For instance, in problems with no predefined matching, the TBS-based routing strategy is suited to solve combination and splitting problems while minimizing the number of movements, but for problems without combination and splitting it might not be the best option. The main reason for that is that although we may have rolling stock of one single type on each parking track, we will still have train units with unordered subtypes on that parking track. This can actually be prevented by using the standard parking rules from the IRTS-based routing strategy, based on ordering trains based on their departure sequence, adapted for problems with no predefined train matching. This can be done by assigning each arriving train to the first departing train with the exact same composition. Since we assume train matching to be injective and surjective, this rule allows us to order all trains according to the departure sequence, and hence solve the matching problem, and to ensure that all trains can depart via non-conflicting routes. If the problem were to have combination and splitting, this strategy would however not be suitable in practice since agents would be ruled by the randomness of the algorithm due to lack of guidance to agents, which generally does not help to solve this problems. Lastly, a hybrid parking strategy is also possible and even desired in cases in which one of the rolling stock types has to solve splitting and combination problems while another rolling stock type in the same problem, but with different subtypes, does not. In this case we could consider using the TBS-based routing strategy for the former and the IRTS-based routing strategy for the latter, with the matching rule described.

Fifthly, the results in Experiment 2 and 3 reveal interesting insights on the overall parking capacity consumption depending on the type of problem and routing strategy. On average, the TBS-RS occupies less parking tracks and their respective lengths are used to a higher extent compared to the IRTS-RS, which we understand as a more efficient use of parking capacity. This is remarkable especially considering that problems using the TBS-RS need one empty parking track to allow resolving conflicts by means of relocation, which the IRTS-RS in general does not need due to the different way of handling relocation movements. In Experiment 3, with a total of 7 train units in the shunting yard, the average number of parking tracks occupied ranges from about 70 to 80%, whereas the average track length occupied ranges from 50 to 60%, which suggests that both RS are potentially scalable to a higher number of train units being handled by the shunting yard. In Experiment 4, by contrast, as the parking capacity is more congested, we observe less differences between the TBS and IRTS-RS on how the parking capacity is used (both about 90-95% of the parking tracks used, at 70% of its length each).

Sixthly, with respect to the number of movements, we have showed that the TBS-RS and the IRTS-RS in general significantly decreased the number of movements required to solve the different types of problems considered in the experiments in TORS-MATDRL compared to the original settings in the programme. The results with RS show average numbers of entry and exit movements close to the expected values as expected based on the problem design (i.e. number of arriving and departing trains), which is a good sign of the performance of the RS in terms of movements. The number of relocation and repositioning movements are reduced to more than half when adding the RS. When comparing with HIP, it is difficult to retrieve conclusions for the case studies carried out in the case study of Heerlen since HIP often leaves trains standing still on the entry track whereas TORS-MATDRL generally moves all trains inwards the shunting yard. As a result, the average number of movements in Heerlen is significantly lower in HIP compared to TORS-MATDRL. For instance, in Experiment 3, with 7 trains units, the average total number of movements ranges between 16.5 and 18.5 with TORS-MATDRL and around 12.5 in HIP, which is up to 32% less. Each train unit entering the shunting yard will perform between 2 and 3 movements on average.

In Kleine Binckhorst, we observe a different behaviour on how movements are handled in TORS-MATDRL and HIP. The RS in TORS-MATDRL prompts agents to move in a clockwise direction, (i.e. closing the *carrousel* circle) in order to start filling in the parking tracks $\tau \in T_p$ as much as possible. As trains are arriving and park on the parking tracks $\tau \in T_p$, once it becomes no longer possible for an arriving train to park on these tracks without violating the standard parking rules, we will this train to violate the clockwise flow pattern and move instead to the entry tracks $\tau \in T_e$, usually from the facility tracks $\tau \in T_s$. Therefore, in short, both the TBS and IRTS-RS prioritize filling in the parking tracks $\tau \in T_p$ as much as possible before using the entry tracks $\tau \in T_e$ for parking. HIP, by contrast, seems to use both the entry tracks $\tau \in T_e$ and parking tracks $\tau \in T_p$ for parking with no apparent priorities. This behaviour works better if the goal is to minimize movements, since closing the circle via entry tracks $\tau \in T_e$ and relocation track $\tau \in T_r$ as the RS stimulate involves more movements than simply parking a train on the empty tracks $\tau \in T_e$ right after arrival or once servicing on the facility tracks $\tau \in T_s$ is completed. Additionally, the average duration of parking movements, which in principle we aim at maximizing, is generally longer in HIP compared to TORS-MATDRL for all experiments. The reason for this is the action triggers design that TORS-MATDRL uses, which in certain cases may not be optimal as certain actions are delayed to later action triggers, which eventually results in shorter parking durations.

Seven, regarding the adaptability and performance of the routing strategies to solve problems in different shunting yard types, the experiments carried out also provide us with some insightful points of discussion. The main finding from the computational point of view is that the problems in Experiment 4 (i.e. Kleine Binckhorst) are more constrained compared to the previous experiments mainly due to the train flow pattern constraints and the bottleneck in the relocation track that agents found difficult to resolve. As a result, the solvability achieved is significantly lower than in the case study of the previous experiments without flow pattern constraints (up to 15% compared to 30-55% in the latter case), with a much longer training time. The case study without flow pattern constraints (i.e. Heerlen) is by contrast more relaxed as agents have much more freedom to move around the shunting yard and use tracks in both directions. Also, the use of the relocation track, which appears to be a source of bottlenecks in the TORS-MATDRL framework with our RS, is much lower in Heerlen since it is only used for conflict resolution.

Eight, in the fourth experiment (section 8.6) we have included both combination and splitting. We designed a list of arriving and departing trains that creates the need for splitting and the combination for rolling stock of the same type. When predefined train matching is given (i.e. with train unit IDs), there is at most one train unit involved in common for both operations (otherwise if we were to have two units in common then we would not have the need to solve combination and splitting subproblems). It is however interesting to note that the variability in the instances generated create situations in which both the splitting and the combination involve independent pairs of train units, while in other cases there is one unit ID in common for both operations. In this case, the corresponding agent has to learn to solve them sequentially (first splitting, then combination). The RS developed give guidance to agents to carry out the individual operations of combination and splitting and movements in between, if necessary.

Lastly, with regards to service tasks, in both case study shunting yards, i.e. *carrousel* without flow pattern constraints (Heerlen) and *carrousel* with flow pattern constraints (Kleine Binckhorst), dedicated service facilities can be accessed from two tracks (33 and 34 in Heerlen and 61 and 62 in Kleine Binckhorst), which can be used simultaneously. Also, servicing on the same track to more than one train is also allowed as long as the trains to be serviced can both physically fit in the track length dedicated to servicing. Trains with associated service tasks are prompted to use them as soon as possible upon arrival, with higher priority than parking on parking tracks. In the case of Heerlen, trains generally clear the service facility by parking on the same track (33 or 34) but in the area not served by the servicing platform (see Figure 7.1), but the service facility itself can also be used for parking. In the case of Kleine Binckhorst, all trains in general have to pass through the service facility tracks (11 or 12) (see Figure 7.3) regardless of whether they have associated service tasks or not. Nevertheless, our point is that there is no guidance in our routing strategies on whether trains should prefer which of the tracks served by the service facility based on certain conditions. These conditions could be whether there are already trains using part of the track capacity and what attributes they have in the current state. The particular attributes of interest would probably depend on the shunting yard type.

On one hand, in the case of Heerlen, we observed in Experiment 3 (section 8.5) with the TBS-RS that train units of different rolling stock type may end up being serviced on the same track connected to the dedicated service facility. This violates the standard parking rules although we allow for that during servicing (we simply do not give a negative reward for that). Nevertheless, since this track will be used just for parking once all service tasks are completed, we will inevitably end up having a parking violation. This could perhaps be prevented by simply sending the train unit of different type to the other service facility track. The violation occurred however does not necessarily mean the instance cannot be solved anymore, it will highly depend on the overall instance and the current state whether it can be solved or not. Similarly, in the IRTS-RS we could also consider giving some more information to agents on the current state to anticipate possible parking violations and hence minimize conflicts. Lastly, servicing in Heerlen is also the source of most repositioning movements, which the RS stimulates in order to clear the service facilities. Nevertheless, some of these repositioning movements are possibly unnecessary and could be prevented. As a result, the average total number of movements would be lower.

On the other hand, in the case of Kleine Binckhorst, the two service facility tracks are used in one single direction and all trains moving to the relocation track have to use them, so service on those tracks might create a bottleneck. Also, we are interested in sending trains to the relocation track to reverse their direction as soon as possible so that it can be cleared as soon as possible. Therefore, we may have preferences on how to manage the capacity of tracks 11 and 12. For instance, if only one of the tracks is being used for service by one train, another arriving train without service tasks associated may prefer to move via the empty track directly to the relocation track instead of waiting behind the train being serviced. It is often the case as well that another train also with service tasks arrives when there is already one being serviced. In this case we may also have preferences on what track to select depending on the time these trains will be ready to continue moving. It is important to note that all these suggestions for both case studies can be implemented but will result in more constrained problems, which can have counter effects in the learning process and the resulting performance. In this work we prompt agents to move to service as soon as possible but beyond this rule we have given freedom to agents to explore what are the best ways to manage the capacity of service facilities. It should be noted that the action space in TORS-MATDRL masks out all but one action *Service*, associated to one specific location. It is possible to include as many *Service* actions as available locations for servicing in the shunting yard, but this would result in a larger action space and hence a higher computational cost.

It should be noted that service tasks are included in all the experiments carried out since servicing is usually present in real-life cases. Removing those would likely make the problem generally simpler from the computational point of view since the environment would be less constrained in terms of train movements and certain situations that can result in a violation would not take place anymore. Similarly, problems without combination or splitting would also be significantly simplified.

8.8. Answer to Sub-Question 4

The experiments carried out in this Chapter allow us to answer the fourth and last research sub question: **What is the performance of the new framework?** Based on the experiments carried out, the performance can be assessed in different dimensions, such as based on the original TORS-MATDRL framework, based different matching problems, based on the routing strategies used, based on different shunting yards, based on the performance metrics and based on HIP and based on other approaches such as classic mathematical optimization.

In the first experiment we have demonstrated that both the TBS and IRTS-based routing strategies significantly improved the computational performance of the learning process of TORS-MATDRL with the original reward heuristics (i.e. without RS) and with the heuristic for random exploration developed in this research. In particular, for problems with 6 train units, the RS have increased the solvability up to 45-55% compared to less than 10% without RS in comparable training times. Our routing strategies successfully stimulate agents to do the right sequences of actions and hence to learn and converge to a policy that can effectively solve instances. A clear improvement is also observed in the performance metrics, in which the produced route plans are much more efficient in both the problems with 0% matching and 100% matching. For instance, the overall number of movements decreased up to 25% and the robustness against disturbances in service tasks is improved as the parking events have average durations from 1.8 to 2.2 times longer, which is also a good sign of having much less unnecessary movements.

The experiments carried out have also shown relevant computational differences between the problems with no matching (i.e. with TBS-RS) and the problems with predefined matching (i.e. with IRTS). For instance, in the problems with 6 train units and no matching, the solvability achieved is about 56% compared to 44.5% in the problem with perfect matching. The reason behind this is that the former is a more relaxed problem in which agents can explore many different options to solve combination problems, which also explains the longer training time, which is 26.8 hours for 0% matching and 13.8 hours for 100% matching. By contrast, with 100% matching, agents only have one possible option to solve each combination problem and hence the problem is more constrained and the algorithm seems to converge faster. It is however necessary to run the algorithms until convergence to make fair comparisons and to retrieve solid conclusions in terms of computational performance. Besides, when the problem complexity increases, in terms of the amount of train units handled in the shunting yard, the problem becomes more constrained and this is reflected by the lower solvability achieved by both RS in more complex problems. For instance, in Experiment 3 the solvabilities achieves are about 32% compared to 45-55% in Experiment 2.

Experiments 2 and 3 have shown interesting insights into the differences between the TBS and the IRTS-based routing strategies, especially concerning the parking capacity consumption. The TBS-RS is used for problems with 0% matching, usually for shunting yards capacity assessment, whereas the IRTS-RS is used for planning of daily operations. The results with 6 and 7 train units have shown that on average, 70% of the parking tracks are used simultaneously on average and those tracks are used up to 45-55% (depending on the number of trains in the instance) of its length with the TBS-RS. By contrast, with the IRTS-RS, around 80% of the parking tracks are used simultaneously on average and those tracks are used up to 35-50% of its length. This is mainly an effect of the standard parking rules, which are more relaxed for the TBS-RS compared to the IRTS-RS. Therefore, the use of parking capacity is more efficient with the TBS-RS since it gives more room to scale up the problem. In any case, the parking capacity consumption with both RS in our experiments in Heerlen show that the problems tackled, with up to 7 train units, are potentially scalable to a higher amount of trains.

In terms of the performance of the routing strategies to solve problems in different shunting yard types, the main finding from the computational point of view is that the problems in Experiment 4 (i.e. Kleine Binckhorst) are more constrained compared to the previous experiments mainly due to the train flow pattern constraints and the bottleneck in the relocation track that agents found difficult to resolve. As a result, the solvability achieved is significantly lower than in the case study of the previous experiments without flow pattern constraints (up to 15% compared to 30-55% in the latter case), with a much longer training time. The case study without flow pattern constraints (i.e. Heerlen) is by contrast more relaxed as agents have much more freedom to move around the shunting yard and to use tracks in both directions. Also, the use of the relocation track, which appears to be a source of bottlenecks in the TORS-MATDRL framework with our RS, is much lower in Heerlen since it is only used for conflict resolution.

In terms of performance metrics, one of the main goals has been to minimize the number of movements. Experiment 1 has revealed that the RS are able to reduce the total number of movements up to 25% compared to TORS-MATDRL without the RS (but with the heuristic for random exploration) for problems with up to 6 train units. It should be noted that it is difficult to quantify the improvement when making these comparisons when one of the approaches often does not move some of the trains in the instance inwards the shunting yard, which often happens in TORS-MATDRL without RS and with HIP in Heerlen. Consequently, for instance, the improvement that can be retrieved from Table 8.9 is in practice significantly higher than 5% in the problems with 0% matching and 25% in the problems with 100% matching. The average number of movements per train unit using the RS in Heerlen ranges between 2.1 and 2.6 movements for problems with 6 and 7 train units.

The experiment carried out in Kleine Binckhorst reveals that the routing strategies developed are not the most efficient to minimize the number of trains movements in problems with 9 train units since HIP finds a different way of handling train movements which involves less movements. With the RS, the average number of movements per train unit ranges between 4.4 and 4.9, whereas it is about 3.0 in the results obtained with HIP. We cannot guarantee however that this will also hold for scaled-up problems, which possibly will not since a more congested infrastructure may lead to a different way of moving and parking trains in HIP to better exploit parking capacity. The logic behind the RS implemented in Kleine Binckhorst already tries to maximize the parking capacity by starting filling in the parking tracks $\tau \in T_p$ (52 to 55) and the rest of parking tracks only once the former can no longer be used without violating the standard parking rules. Therefore, this strategy seems to penalize the number of movements.

Our heuristic for random exploration and routing strategies developed and implemented in TORS-MATDRL are found to be a promising alternative to other approaches to the TUSP such as mathematical optimization. Based on the literature review carried out, mathematical optimization approaches are computationally expensive and very limited applicability in real-life cases, in terms of the amount of train units being handled and the ability to incorporate all the TUSP subproblems. TORS-MATDRL with the heuristic for random exploration and routing strategies has proved to be able to solve problems with up to 9 train units (and potentially more) with all TUSP subproblems except of crew scheduling, which has not been included in this research for simplicity, but TORS-MATDRL is flexible enough to incorporate it as well. Training time in the case of Experiment 4 took more than 72 hours without clearly reaching convergence. Therefore, training time is obviously one of the downsides of using artificial intelligence approaches. Nonetheless, on the positive side, once the appropriate models are trained, the process of inference of results is extremely fast and it is much easier to understand the process of inference of results, which delivers more consistent results.

9 Conclusions and Recommendations

This chapter presents the conclusions (section 9.1) and recommendations (section 9.2) of this research.

9.1. Conclusions

Research sub question 1: What information from the logistics side of routing in the Train Unit Shunting Problem can be incorporated into the current multi-agent deep reinforcement learning framework developed by NS? We understand this information as routing strategies, in order to minimize the number of train movements and hence to help improving route plans while taking into account the characteristics of each shunting yard type and the type of matching subproblem. This is motivated by the work by Beerthuizen (2018), in which policies from other logistics problems were successfully applied in the TUSP in a different framework. Furthermore, the research on rewards shaping for RL reveal the complexity of designing reward functions such that lead agents to a goal (Dewey, 2014) and there are limited insights on how to reduce the search space for agents by means of these functions (Matignon et al., 2006), which play a significant role in speeding up the learning process of agents (Goyal et al., 2019).

The scientific gap researched is the development of routing strategies such that take all the above mentioned considerations into account and to formulate them in the multi-agent DRL framework developed by NS (NS, 2020). At the higher level, the scientific gap concerns the incorporation of information from the logistics side of routing in the TUSP into a multi-agent DRL framework in order to optimize the output route plans and to contribute towards a better learning process by helping agents to make the right decisions from the logistics efficiency point of view.

In relation to the previous, there is also lack of insights on the adaptability and scalability of such routing strategies to fit different problems, such as different shunting yard types or matching problems. This information can be useful for the railway operation point of view since it can explain how generic and flexible can routing strategies be to fit in different train shunting problems. Besides, from the multi-agent DRL framework point of view, it can explain what are the limitations of the framework and what is its applicability to solve other routing problems using DRL.

Research sub question 2: How can routing strategies be formulated in the multi-agent deep reinforcement learning framework in order to produce optimal route plans and to speed up learning? The routing optimization solution consists of Heuristics for Random Exploration and the Heuristics for Routing Strategies, which include a TBS-based routing strategy and an IRTS-based routing strategy. Firstly, the heuristics for random exploration based on the principles of large neighbourhood search is required to efficiently search the action space, in which the distribution and importance of actions is highly imbalanced. Additionally, these heuristics significantly help speeding up the learning process, which by pure random exploration would be computationally very expensive.

Secondly, two routing strategies are developed and implemented based on the parking strategies introduced by Beerthuizen (2018), which are the Type-Based Strategy (TBS) and the In-Residence Time Strategy (IRTS) respectively, which we extend in this work to include complete routing strategies consisting of four components: (1) standard parking rules, (2) combination and split rules, (3) conflict resolution rules and (4) unnecessary movements rules. The routing strategies are formulated as reward functions in the TORS-MATDRL framework. The resulting routing strategies are named after the parking strategies, i.e. TBS and IRTS-based routing strategies. The reward functions try to give rewards as much as possible and are carefully shaped to ensure a hierarchy in the sequence of decisions and events that agents have to make to complete an instance correctly. Lastly, the routing strategies are designed in a way so that they are scalable but have certain limitations in their implementations.

Research sub question 3: How flexible and scalable are the routing strategies in different shunting yards?
The adaptation and formulation of the routing strategies developed to different case studies as described in this Section allows us to answer the third research sub question. In chapter 6 we have developed on one hand an heuristic for random exploration, which are deemed to be applicable for any kinds of shunting yards since these focus on particular sequences of actions that we want to stimulate agents to learn, such as setting back instead of waiting, clearing the gateway track, departing, starting a service task if there are pending service tasks, or performing a correct combination or split.

On the other hand, we have developed the TBS and IRTS-based routing strategies for a generic shunting yard consisting of a gateway track $\tau \in T_g$, a set of parallel parking tracks T_p, and a relocation track $\tau \in T_r$ on the opposite side of the gateway track, in such a way that all parking tracks can be reached from both the gateway track and the relocation track without reversing the train direction. Each routing strategy consists of four components, which are (1) standard parking rules, (2) combination and split rules, (3) conflict resolution rules and (4) unnecessary movements rules. These rules can be applied at any type of shunting yard but need specific adjustments to fit each type (e.g. *carrousel*, shuffleboard), and further adjustments to fit the particular characteristics of real-life shunting yards.

Firstly, for the *carrousel* type of shunting yard without flow pattern constraints, the routing strategies as described in section 6.2 can be applied directly to such case. Secondly, for the case of *carrousel* shunting yards with flow pattern constraints, extensions to stimulate agents to learn to follow a predefined flow pattern are required, as well as further extensions to let agents violate this flow in particular cases for conflict resolution. For this, a distinction between groups of parking tracks can be made, possibly including entry tracks T_e or exit tracks T_x on top of normal parking tracks T_p. Entry or exit tracks may be used for temporary parking or sorting, whereas normal parking tracks are used preferably for long parking and train sorting. Also, when parking tracks are used in single directions it means that in general, most trains will have to reverse their direction on a relocation track $\tau \in T_r$ in order to close the characteristic circle of *carrousel* shunting yards. The relocation track might therefore become a bottleneck and hence we may prefer to speed up all activities carried out upstream or move them downstream as much as possible, such as splitting and combination operations. Therefore, the environment in this case is generally more constrained compared to the case without such flow constraints. In terms of conflict resolution, since parking tracks are used in one single direction, the preconditions for conflict resolution are also different compared to the case without flow constraints, which slightly modifies for instance the way relocation is handled.

The real-life shunting yards considered for the experiments in this research have further particularities that have to be taken into account in the adaptations of the routing strategies, such as the location of service facilities and how these are connected to parking tracks and other tracks.

Lastly, further adaptations of the routing strategies as described in section 6.2 are needed for the shuffleboard type of shunting yards and for hybrid or other unclassified shunting yards. The main difference in shuffleboard shunting yards is that parking tracks $\tau \in T_p$ are LIFO tracks. The standard parking rules are applied as normally (but only from the open side of parking tracks), as well as combination and split rules and unnecessary movements rules. Nonetheless, conflict resolution rules again need adaptations to fit this particular problem design. The preconditions to trigger relocation movements stay the same but the way train movements are handled is quite different since only the gateway track side of the parking tracks $\tau \in T_p$ is open.

In conclusion, the routing strategies described in section 6.2 are flexible enough to be applied to various types of shunting yards. The fundamental four components of each routing strategy are generally replicable to various contexts and further extensions and constraints may be added to fit particular problem designs. Lastly, we have shown how this can be formulated in a multi-agent DRL framework.

Research sub question 4: What is the performance of the new framework? Based on the experiments carried out, the performance can be assessed in different dimensions, such as based on the original TORS-MATDRL framework, based different matching problems, based on the routing strategies used, based on different shunting yards, based on the performance metrics and based on HIP and based on other approaches such as classic mathematical optimization.

In the first experiment we have demonstrated that both the TBS and IRTS-based routing strategies significantly improved the computational performance of the learning process of TORS-MATDRL with the original reward heuristics (i.e. without RS) and with the heuristic for random exploration developed in this research. In particular, for problems with 6 train units, the RS have increased the solvability up to 45-55% compared to less than 10% without RS in comparable training times. Our routing strategies successfully stimulate agents to do the right sequences of actions and hence to learn and converge to a policy that can effectively solve instances. A clear improvement is also observed in the performance metrics, in which the produced route plans are much more efficient in both the problems with 0% matching and 100% matching. For instance, the overall number of movements decreased up to 25% and the robustness against disturbances in service tasks is improved as the parking events have average durations from 1.8 to 2.2 times longer, which is also a good sign of having much less unnecessary movements.

The experiments carried out have also shown relevant computational differences between the problems with no matching (i.e. with TBS-RS) and the problems with predefined matching (i.e. with IRTS). For instance, in the problems with 6 train units and no matching, the solvability achieved is about 56% compared to 44.5% in the problem with perfect matching. The reason behind this is that the former is a more relaxed problem in which agents can explore many different options to solve combination problems, which also explains the longer training time, which is 26.8 hours for 0% matching and 13.8 hours for 100% matching. By contrast, with 100% matching, agents only have one possible option to solve each combination problem and hence the problem is more constrained and the algorithm seems to converge faster. It is however necessary to run the algorithms until convergence to make fair comparisons and to retrieve solid conclusions in terms of computational performance. Besides, when the problem complexity increases, in terms of the amount of train units handled in the shunting yard, the problem becomes more constrained and this is reflected by the lower solvability achieved by both RS in more complex problems. For instance, in Experiment 3 the solvabilities achieves are about 32% compared to 45-55% in Experiment 2.

Experiments 2 and 3 have shown interesting insights into the differences between the TBS and the IRTS-based routing strategies, especially concerning the parking capacity consumption. The TBS-RS is used for problems with 0% matching, usually for shunting yards capacity assessment, whereas the IRTS-RS is used for planning of daily operations. The results with 6 and 7 train units have shown that on average, 70% of the parking tracks are used simultaneously on average and those tracks are used up to 45-55% (depending on the number of trains in the instance) of its length with the TBS-RS. By contrast, with the IRTS-RS, around 80% of the parking tracks are used simultaneously on average and those tracks are used up to 35-50% of its length. This is mainly an effect of the standard parking rules, which are more relaxed for the TBS-RS compared to the IRTS-RS. Therefore, the use of parking capacity is more efficient with the TBS-RS since it gives more room to scale up the problem. In any case, the parking capacity consumption with both RS in our experiments in Heerlen show that the problems tackled, with up to 7 train units, are potentially scalable to a higher amount of trains.

In terms of the performance of the routing strategies to solve problems in different shunting yard types, the main finding from the computational point of view is that the problems in Experiment 4 (i.e. Kleine Binckhorst) are more constrained compared to the previous experiments mainly due to the train flow pattern constraints and the bottleneck in the relocation track that agents found difficult to resolve. As a result, the solvability achieved is significantly lower than in the case study of the previous experiments without flow pattern constraints (up to 15% compared to 30-55% in the latter case), with a much longer training time. The case study without flow pattern constraints (i.e. Heerlen) is by contrast more relaxed as agents have much more freedom to move around the shunting yard and to use tracks in both directions. Also, the use of the relocation track, which appears to be a source of bottlenecks in the TORS-MATDRL framework with our RS, is much lower in Heerlen since it is only used for conflict resolution.

In terms of performance metrics, one of the main goals has been to minimize the number of movements. Experiment 1 has revealed that the RS are able to reduce the total number of movements up to 25% compared to TORS-MATDRL without the RS (but with the heuristic for random exploration) for problems with up to 6 train units. It should be noted that it is difficult to quantify the improvement when making these comparisons when one of the approaches often does not move some of the trains in the instance inwards the shunting yard, which often happens in TORS-MATDRL without RS and with HIP in Heerlen. Consequently, for instance, the improvement that can be retrieved from Table 8.9 is in practice significantly higher than 5% in the problems with 0% matching and 25% in the problems with 100% matching. The average number of movements per train unit using the RS in Heerlen ranges between 2.1 and 2.6 movements for problems with 6 and 7 train units.

The experiment carried out in Kleine Binckhorst reveals that the routing strategies developed are not the most efficient to minimize the number of trains movements in problems with 9 train units since HIP finds a different way of handling train movements which involves less movements. With the RS, the average number of movements per train unit ranges between 4.4 and 4.9, whereas it is about 3.0 in the results obtained with HIP. We cannot guarantee however that this will also hold for scaled-up problems, which possibly will not since a more congested infrastructure may lead to a different way of moving and parking trains in HIP to better exploit parking capacity. The logic behind the RS implemented in Kleine Binckhorst already tries to maximize the parking capacity by starting filling in the parking tracks $\tau \in T_p$ (52 to 55) and the rest of parking tracks only once the former can no longer be used without violating the standard parking rules. Therefore, this strategy seems to penalize the number of movements.

Our heuristic for random exploration and routing strategies developed and implemented in TORS-MATDRL are found to be a promising alternative to other approaches to the TUSP such as mathematical optimization. Based on the literature review carried out, mathematical optimization approaches are computationally expensive and very limited applicability in real-life cases, in terms of the amount of train units being handled and the ability to incorporate all the TUSP subproblems. TORS-MATDRL with the heuristic for random exploration and routing strategies has proved to be able to solve problems with up to 9 train units (and potentially more) with all TUSP subproblems except of crew scheduling, which has not been included in this research for simplicity, but TORS-MATDRL is flexible enough to incorporate it as well. Training time in the case of Experiment 4 took more than 72 hours without clearly reaching convergence. Therefore, training time is obviously one of the downsides of using artificial intelligence approaches. Nonetheless, on the positive side, once the appropriate models are trained, the process of inference of results is extremely fast and it is much easier to understand the process of inference of results, which delivers more consistent results.

Main research question: How can routing strategies for the Train Unit Shunting Problem in a Multi-Agent Deep Reinforcement Learning framework help optimizing route plans to support railway planners? It can be stated that the heuristic for random exploration and the routing strategies developed and formulated in the multi-agent DRL framework successfully help agents recognizing patterns and performing the right actions to solve the TUSP in different problem designs, such as with either no matching or predefined matching, and in different shunting yard types. Therefore, this approach is found to be effective and efficient to produce route plans in that can support railway planners to solve shunting planning assignments in real-time and also in earlier stages of planning. Nonetheless, due to the long training time, TORS-MATDRL will not be that convenient for instance to assess the effects of certain modifications in the shunting yard track layout for long-term planning. HIP may still be more suitable for such tasks.

TORS-MATDRL, with the extensions developed in this research, i.e. the heuristic for random exploration and the routing strategies, addresses a real-life problem and the solutions produced by this framework have practical effects on problems such as real-time train shunting. Our approach can be applied to a broader range of shunting yards types for any mainline railway, metro or tram network. With appropriate adaptations it is also possible to apply our approach for freight yards. Even broader, we believe that similar approaches with collaborative multi-agent cooperation tasks can be developed to solve other sequential decision making problems in the field of transportation and logistics. Our results demonstrate that DRL has potential to solve routing optimization problems efficiently and effectively. Nevertheless, more research is needed before artificial intelligence can represent a solid alternative to classic operations research approaches to certain problems.

9.2. Recommendations

The recommendations for further research and the practical application of this research is discussed in this Section. First, the Scientific points are presented in subsection 9.2.1. The practical application of this research is described in subsection 9.2.2.

9.2.1. Scientific Recommendations

Firstly, with respect to the learning parameters, we have assumed values backed by previous works such as Barnhoorn (2020) and our own experience working with DRL. Consequently, it would be interesting to carry out some sensitivity analysis to assess the impact of parameters in performance. Nonetheless, since training is a time-consuming process, such a sensitivity analysis would also be very time-consuming.

One of the major challenges faced when working with reinforcement learning is that of rewards shaping. In this kind of sequential decision making problems, with long sequences of actions for an agent, it is advised to give as much rewards as possible since there are many actions and state changes in one instance and this makes the reward very sparse and also very often rewards are delayed. In general, agents learn better when they can receive rewards immediately after they make an action. The main challenge is to make sure whether the reward really reflects what we want the agent to learn, since we get what we incentivize, not what we intend. This difficulty has been observed when defining the reward functions for our routing strategies. Our conclusion is that rewards in general have to be carefully defined for different conditions of the agents and the state with positive rewards for correct actions and negative rewards for incorrect actions to prevent agents from over exploiting correct actions. Also, its values also have to be carefully defined in order to prompt agents to perform the right sequences of tasks.

Additionally, further research and effort is needed to ensure that agents learn to avoid violations. Firstly, the violations associated with trains not being able to leave are generally the most common ones, which usually relate to conflicts or bottlenecks that agents are unable to resolve. The main issue is that despite the routing strategies guide them to perform the required sequence of actions to resolve those conflicts, it is often the case that some of these actions are delayed in time to subsequent action triggers and sometimes too late. The reason for this is that despite the action triggers design and the rewards assigned to agents, the policy learned with training does not match perfectly the expected behaviour, and other agents with higher expected q-values for certain other actions may act instead in a given action trigger. This has been often observed in the Heerlen case studies with agents being unable to resolve conflicts and in Kleine Binckhorst with a remarkable bottleneck in the relocation track. Secondly, violations concerning trains blocking the entry track are generally much less common and hence it is not that much of a major concern. Therefore, we advise to further investigate the behaviour of agents and the influence of the action triggers design to better tackle the described major issues concerning conflicts and bottlenecks. This could potentially have a significant effect on improving performance and hence the quality of the produced route plans.

An alternative methodology to the one developed in this research to enhance the learning of agents with heuristics is to categorize moving actions. This means that, agents are given a single moving action in the action space (among other feasible actions), which is not associated to any destination track and hence will not be learnt by agents. Here, some heuristics or simple rules would decide in a fully deterministic way to what destination track the associated train will move, should the agent decide to choose the Move action, based on the information of the environment perceived by the agent. By doing this, the optimality (or at least close to) of this movement can be guaranteed. Therefore, the action space is adjusted and significantly reduced. This alternative approach may be easier to learn for agents, due to the reduced action space, but the overall computational cost will be higher compared to the approach in this research due to the enhancement with heuristics described.

9.2.2. Practical Recommendations

In this research we have carried out experiments with ad-hoc instances with a maximum of 7 train units in the shunting yard of Heerlen and 9 train units in the shunting yard of Kleine Binckhorst for both the TBS and the IRTS-based routing strategies. During the implementation of these heuristics, we have gradually increased the complexity in terms of the number of train units to this extent, for which the heuristics proved to be scalable. Nonetheless, we recommend to further scale up these problems with more train units to verify the scalability of the developed heuristics and to eventually solve real-life planning assignments. In such scaled-up problems it may also be insightful to check again the performance in terms of number of movements in different shunting yards and problem designs by comparing it to HIP as we have done in this research. Besides, in the case of the shunting yard of Heerlen, it is also recommended to incorporate non-service traffic, which is missing in our experiments, especially if the routing strategies were to address real-life planning assignments for Heerlen.

Additionally, the routing strategies developed have certain limitations for its practical application, some of which are collected in subsection 6.2.4. In the case studies used in this research, these have shown to make a little impact on the results. Nonetheless, some of these may need to be taken into account for the application of the RS in scaled-up problems or in other shunting yard layouts, which are cases in which these limitations might have a greater impact, either with increased number of conflicts or track layout particularities.

Besides, in section 7.4 we have given some guidelines on how to adapt the routing strategies developed for the shuffleboard type of shunting yard and hence we also recommend formulating them in TORS-MATDRL to further demonstrate their capabilities and flexibility to fit different problem designs. For an even broader application of the algorithms, based on the adaptations of the RS for both *carrousel* and shuffleboard types of shunting yards, it is also possible to develop adaptations for hybrid types of shunting yards, also known as station types (Janssens, 2017), and other unclassified types.

It is also advised to use real-life planning assignments in order to test the developed implementations and to further develop them to improve its performance, particularly if TORS-MATDRL were to be used by NS in practice as a software product. Besides, the crew scheduling subproblem of the TUSP should also be incorporated in the testing phase of the implementations in TORS-MATDRL to study their effects on our routing strategies, what adaptations or adjustments are needed to solve consequently more constrained instances and to eventually produce more realistic route plans.

It is important to highlight that the developed routing strategies are based on the current practice at NS but also on the author's own judgement and logic reasoning. The latter has been necessary to make certain decisions and to define certain rules, but it does not necessarily reflect how railway planners would actually solve planning assignments. The reason for this is that in this research we have not included a feedback loop involving planners and that we have not tested our implementations to solve real-life assignments. This feedback loop could be explored in a follow-up research to take into account the input from planners to solve real-life assignments following the methodology proposed by Baurichter (2020) and eventually improve the performance of TORS-MATDRL.

In order to better assess the performance of TORS-MATDRL against other approaches such as HIP, especially when solving real-life assignments, we advise to ensure that both approaches use the exact same configuration and parameters to produce route plans so that the comparison e.g. in terms of movements, which is one of the key points in this research, is fair and meaningful.

In conclusion, we believe that TORS-MATDRL, with the heuristic for random exploration and the routing strategies developed, is a promising alternative to solve the TUSP in real-life planning assignments, although it still requires significant effort for further research and development of the programme. To support that, in this section we have provided a number of advices, guidelines and research directions for NS to continue with this work.

Bibliography

[1] MM. Alipour and SN. Razavi. A new multiagent reinforcement learning algorithm to solve the symmetric traveling salesman problem. *Multiagent and Grid Systems*, 11(2):107–119, 2015.

[2] K. Arulkumaran, MP. Deisenroth, M. Brundage, and AA. Bharath. A brief survey of deep reinforcement learning. *IEEE Signal Processing Magazine, Special Issue on Deep Learning for Image Understanding*, 2017. doi: 10.1109/MSP.2017.2743240.

[3] Q. Barnhoorn. A Multi-Agent Approach to the Train Unit Shunting Problem. Technical report, Bachelor's Thesis in Artificial Intelligence. Radboud University, 2020.

[4] M. Baurichter. Adding human aspects to Planning Assistance Tools for complex logistical processes: a case-study of railway node planning in the Netherlands. Master's thesis, 2020.

[5] E. Beerthuizen. Optimizing train parking and shunting at NS service yards. Master's thesis, Delft, 2018. URL `http://resolver.tudelft.nl/uuid:d5c93539-f182-4d4d-931b-c5356727c827`.

[6] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[7] N. Bešinović and RMP. Goverde. Stable and robust train routing in station areas with balanced infrastructure capacity occupation. *Public Transport*, 11(2):211–236, 2019.

[8] S. Burggraeve and P. Vansteenwegen. Robust routing and timetabling in complex railway stations. *Transportation Research Part B: Methodological*, 101:228–244, 7 2017.

[9] L. Buşoniu, R. Babuška, and B. De Schutter. Multi-agent reinforcement learning: An overview. In *Innovations in multi-agent systems and applications-1*, pages 183–221. Springer, 2010.

[10] B. Chen and J. Su. Addressing reward engineering for deep reinforcement learning on multi-stage task. In *International Conference on Neural Information Processing*, pages 309–317. Springer, 2019.

[11] S. Cicerone, G. D'Angelo, G. D. Stefano, D. Frigioni, and A. Navarra. Recoverable robustness for train shunting problems. *Algorithmic Operations Research*, 4:102–116, 2009.

[12] I. Cohen. A multi-agent deep reinforcement learning approach to the train unit shunting problem. *Jheronimus Academy of Data Science (JADS)*, 2019.

[13] A. D'Ariano, F. Corman, and D. Pacciarelli. Rescheduling. In *Railway Timetabling & Operations*, chapter 13, pages 253–271. DVV Media Group GbmH | Eurailpress, Hamburg, Germany, 2014.

[14] J. Den Ouden. Generating robust schedules for train maintenance staff. Master's thesis, 2018. URL `http://dspace.library.uu.nl/bitstream/handle/1874/362947/thesis.pdf?sequence=2&isAllowed=y`.

[15] J. Den Ouden. The inner workings of HIP. Technical report, NS, Utrecht, 2019.

[16] D. Dewey. Reinforcement learning and the reward engineering principle. In *2014 AAAI Spring Symposium Series*, 2014.

[17] R. Freling, RM. Lentink, LG. Kroon, and D. Huisman. Shunting of passenger train units in a railway station. *Transportation Science*, 39(2):261–272, 2005.

[18] T. Gabel and M. Riedmiller. On a successful application of multi-agent reinforcement learning to operations research benchmarks. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 68–75. IEEE, 2007.

[19] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 315–323, 2011.

[20] A. Gosavi. Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing*, 21(2):178–192, 2009.

[21] RMP. Goverde. Lecture Railway operations II. Technical report, 2018.

[22] RMP. Goverde and IA. Hansen. Performance indicators for railway timetables. *2013 IEEE International Conference on Intelligent Rail Transportation Proceedings*, 2013. doi: 10.1109/icirt.2013.6696312.

[23] P. Goyal, S. Niekum, and RJ. Mooney. Using natural language for reward shaping in reinforcement learning. 2019.

[24] JT. Haahr, RM. Lusby, and JC. Wagenaar. Optimization Methods for the Train Unit Shunting Problem. *European Journal of Operational Research*, 262(3):981–995, 2017.

[25] R. Haijema, C. Duin, and NM. Van Dijk. Train shunting: A practical heuristic inspired by dynamic programming. *Planning in Intelligent Systems: Aspects, Motivations, and Methods*, pages 437–475, 2006.

[26] Y. Hirashima. A new reinforcement learning system for train marshaling with selectable desired layout. *IFAC Proceedings Volumes*, 44(1):6976–6981, 2011.

[27] Y. Hirashima. A reinforcement learning for train marshaling based on the processing time considering group layout of freight cars. In *Lecture Notes in Engineering and Computer Science: Proceedings of The International MultiConference of Engineers and Computer Scientists 2012, IMECS 2012, 14-16 March, 2012, Hong Kong*, pages 47–52, 2012.

[28] J. Holler, R. Vuorio, Z. Qin, X. Tang, Y. Jiao, T. Jin, S. Singh, C. Wang, and J. Ye. Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. In *2019 IEEE International Conference on Data Mining (ICDM)*, 2019. doi: 10.1109/ICDM.2019.00129.

[29] H. Jamshidi. Application of Deep Learning in Train Shunting Problem. Technical report, Delft University of Technology and NS, 2019.

[30] S.E.B. Janssens. Empirical Analysis of Service Locations at NedTrain. A closer look at the stabling and handling capacity. Master's thesis, Delft, 2017. URL `https://repository.tudelft.nl/islandora/object/uuid%3Af50c79b5-ee5b-4016-b9db-f075dd8bf417`.

[31] LP. Kaelbling, ML. Littman, and AW. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[32] H. Khadilkar. A scalable reinforcement learning algorithm for scheduling railway lines. *IEEE Transactions on Intelligent Transportation Systems*, 20(2):727–736, 2018.

[33] DP. Kingma and JL. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations ICLR*, 2015.

[34] E. Kleine. Generating robust solutions for the Train Unit Shunting Problem under uncertainty: A Local Search based approach. Master's thesis, Eindhoven, 2019. URL `https://research.tue.nl/files/140069333/Master_Thesis_Esmee_Kleine.pdf`.

[35] W. Kool, H. Van Hoof, and M. Welling. Attention, learn to solve routing problems! *ICLR 2019*, 2018.

[36] LG. Kroon, RM. Lentink, and A. Schrijver. Shunting of passenger train units: An integrated approach. *Transportation Science*, 42(4):436–449, 2008.

[37] G. Kyziridis. Deep Reinforcement Learning Algorithms for the Train Unit Shunting Problem. Master's thesis, 2019.

[38] RM. Lentink. Algorithmic Decision Support for Shunt Planning. Technical report, Erasmus University Rotterdam, 2006.

[39] RM. Lentink, PJ. Fioole, LG. Kroon, and C. Van't Woudt. Applying Operations Research techniques to planning of train shunting. Technical report, Erasmus University, Rotterdam, 2003.

[40] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56, 2016.

[41] L. Matignon, GJ. Laurent, and N. Le Fort-Piat. Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning. In *International Conference on Artificial Neural Networks*, pages 840–849. Springer, 2006.

[42] S. Mohanty, E. Nygren, F. Laurent, M. Schneider, C. Scheller, N. Bhattacharya, J. Watson, A. Egli, C. Eichenberger, C. Baumberger, G. Vienken, I. Sturm, G. Sartoretti, and G. Spigler. Flatland-rl: Multi-agent reinforcement learning on trains, 2020. URL `https://arxiv.org/abs/2012.05893`.

[43] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pages 9839–9849, 2018.

[44] L. Ning, Y. Li, M. Zhou, H. Song, and H. Dong. A deep reinforcement learning approach to high-speed train timetable rescheduling under disturbances. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3469–3474, 2019.

[45] NS. Annual Report 2015, 2016.

[46] NS. Functionele beschrijving Opstel Plan Generator (OPG). Technical report, NS, 2018. URL `www.ortec.com`.

[47] NS. TORS and MATDRL, 2020.

[48] NS. Instance Generator. Technical report, NS Knooppunt Logistiek, 2020.

[49] E. Nygren, A. Egli, G. Spigler, M. Ljungström, J. Watson, C. Eichenberger, G. Mollard, and S. Mohanty. Flatland challenge: Multi agent reinforcement learning on trains. 2019. doi: 10.13140/RG.2.2.24477.05601.

[50] E. Peer, V. Menkovski, Y. Zhang, and WJ. Lee. Shunting trains with deep reinforcement learning. *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2018. doi: 10.1109/SMC.2018.00520.

[51] WB. Powell. A unified framework for stochastic optimization. *European Journal of Operational Research*, 275(3):795–821, 2019.

[52] WB. Powell. *Reinforcement Learning and Stochastic Optimization. A Unified Framework for Sequential Decisions.* John Wiley & Sons, Inc, 2020.

[53] ProRail. Annual Report 2016, 2016.

[54] ProRail. Netverklaring 2020. Technical report, Utrecht, 2019.

[55] ML. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, 2005.

[56] Z. Qin, J. Tang, and J. Ye. Deep reinforcement learning with applications in transportation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3201–3202, 2019.

[57] A. Radtke. Infrastructure Modelling. In *Railway Timetabling & Operations*, chapter 3, pages 47–63. DVV Media Group GbmH | Eurailpress, Hamburg, Germany, 2014.

[58] A. Sharpanskykh. Learning in Multi-Agent Systems. Technical report, 2019.

[59] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming - CP982014*, pages 417–431. Springer, 1998.

[60] E. Solinen, G. Nicholson, and A. Peterson. A microscopic evaluation of railway timetable robustness and critical points. *Journal of Rail Transport Planning & Management*, pages 207–223, 2017.

[61] Sporenplan Online. Sporenplan Online, 2019. URL `http://www.sporenplan.nl`.

[62] P. Toth and D. Vigo. *The Vehicle Routing Problem*. Università degli Studi di Bologna. Society for Industrial and Applied Mathematics, 2002. ISBN 978-0-89871-498-2.

[63] Treinposities. Treinposities, 2021. URL `https://treinposities.nl/`.

[64] RW. Van Den Broek. Train Shunting and Service Scheduling: an Integrated Local Search Approach. Master's thesis, Utrecht, 2016.

[65] S. Wang, W. Chaovalitwongse, and R. Babuska. Machine learning algorithms in bipedal robot control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 42:728–743, 2012.

[66] C. White. Interlocking principles. *IET Professional Development Course on Railway Signalling and Control Systems (RSCS 2012)*, pages 75–88, 2012.

[67] F. Wolfhagen. The train unit shunting problem with reallocation. Master's thesis, Erasmus University Rotterdam, 2017. URL `https://thesis.eur.nl/pub/37696`.

[68] Q. Zhong, Y. Zhang, D. Wang, Q. Zhong, C. Wen, and Q. Peng. A mixed integer linear programming model for rolling stock deadhead routing before the operation period in an urban rail transit line. *Journal of Advanced Transportation*, 2020(3), 2020. doi: 10.1155/2020/3809734.

[69] G. Zuo, Q. Zhao, J. Lu, and J. Li. Efficient hindsight reinforcement learning using demonstrations for robotic tasks with sparse rewards. *International Journal of Advanced Robotic Systems*, 17(1):1–13, 2020.

[70] PJ. Zwaneveld, LG. Kroon, HE. Romeijn, M. Salomon, S. Dauzere-Peres, SPM. Van Hoesel, and HW. Ambergen. Routing trains through railway stations: Model formulation and algorithms. *Transportation science*, 30(3):181–194, 1996.

A
Scientific Paper

The scientific paper herein presented discusses part of the complete research that has been conducted throughout this master thesis.

Routing Optimization for the Train Unit Shunting Problem in a Multi-Agent Deep Reinforcement Learning Framework

J. Trepat Borecka[a,*], N. Bešinović[a], Y.M. Maknoon[b], R.M.P. Goverde[a], W. Lee[c]

[a] Delft University of Technology, Faculty of Civil Engineering and Geosciences, 2628 CN Delft, The Netherlands
[b] Delft University of Technology, Faculty of Technology, Policy and Management, 2628 BX Delft, The Netherlands
[c] Nederlandse Spoorwegen, Research & Development Node Logistics, 3511 ER Utrecht, the Netherlands

Abstract—**In busy passenger railway networks, a large amount of trains have to be parked in shunting yards off the mainline every night, where they will be cleaned, maintained, sorted and parked. This problem is known as the Train Unit Shunting Problem (TUSP), which is a hard combinatorial optimization problem faced by railway operators. The TUSP is currently solved using human heuristics, which is difficult and time-consuming. Reinforcement learning approaches have been developed in the last few years to efficiently approach this problem. In this research we develop, in a multi-agent deep reinforcement learning framework, an heuristic for random exploration to efficiently search the state-action space and two heuristics for train routing strategies, which aim at improving the performance and quality of the produced route plans. On one hand, we develop the Type-Based Routing Strategy, based on the idea of parking trains of the same rolling stock type on the same tracks. On the other hand, we develop the In-Residence Time Routing Strategy, based on parking trains ordered according to their departure time. Both routing strategies consists of four components: (1) standard parking rules, (2) combination and split rules, (3) conflict resolution rules and (4) unnecessary movements rules. The goal is to incorporate information from the logistics side of the problem into the framework. We demonstrated the performance of the heuristics developed in two real-life cases in the Dutch railway network. The results of the experiments carried out demonstrate the potential of the resulting framework to produce more efficient route plans and to adapt to different problems designs such as different matching problems types and different shunting yards.**

Keywords: train unit shunting problem, passenger railway optimization, multi-agent systems, deep reinforcement learning, routing, scheduling.

I. Introduction

In busy passenger railway networks, a large amount of trains have to be parked in shunting yards during night time. Shunting yards are sets of parallel tracks for sorting and storing trains off the mainline so that the traffic on the mainline is not obstructed. For instance, the *Nederlandse Spoorwegen* (NS) is the largest passenger railway operator in the Netherlands, which manages a fleet of around 600 train units on 3,434 kilometres of rails, transporting up to 1.2 million passengers per day. A train unit is a self-propelled composition of one or multiple carriages in a fixed order. Each train can consist of one or multiple train units. In order to guarantee the high level of service expected by passengers, trains need to be cleaned, maintained and depart on time to start operating new train services. Train timetables are generally strongly linked with the operations in shunting yards, in the sense that the operating capacity of shunting yards is often the limiting factor towards dispatching a higher number of train services to the network, especially in busy railway networks. As a result, there is an increasing interest to study the train shunting processes and the operational capacity of shunting yards and how it can be maximized.

The associated activities in shunting yards, which include parking, cleaning, maintenance tasks, coupling and splitting of trains are known as a whole as the Train Unit Shunting Problem (TUSP). The TUSP has been traditionally solved by hand by railway planners, who apply human heuristics to solve it. However, this planning task is often difficult and time-consuming for humans. It is important to note that routing strongly depends on the track layout in the yard (i.e. the amount of parking tracks, connections between these and with service facilities, etc.). In practice, each yard is quite unique, but there are certain identifiable characteristics of the infrastructure layout that allow us to classify shunting yards into different types. Besides, routing is also influenced by the matching between arriving and departing trains, which can either be given or needs to be solved otherwise.

With the gradual increase in passenger ridership, the commissioning of new trains is required to expand train fleets and to operate more services. Subsequently, the planning of shunting yard activities will become more and more challenging due to the increased railway traffic and the fact that in many networks the available infrastructure is reaching its capacity limits. One option to solve these infrastructure bottlenecks is to expand the infrastructure at shunting yards. Nevertheless, the commissioning of new infrastructure is costly and the physical room for expansion is often very limited. Consequently, in order to increase infrastructure capacity, it is essential to explore how to optimize the usage of the already existing infrastructure.

Artificial intelligence approaches have been found to be powerful to solve some challenging problems in the field of operations research the past few years [1] [23] [21] [12]. One of the motivations to use deep reinforcement learning (DRL) in the TUSP is the consistency of its outputs for similar problems [29], which is appreciated by planners for the acceptance of Planning Assistance Tools. This feature of DRL is not necessarily the case for classic optimization approaches and local search heuristics [29]. Planning of tasks at the operational and real-time planning level such as train shunting are performed on a daily basis, for which solution algorithms must be very fast. This is possible by using DRL, since a DRL-based model can be trained in advance and for its use it simply has to perform inference of results in a specific real-life problem. Some further promising developments of DRL for the TUSP can be found, e.g. [13] [4] [18] [2] [26].

This paper develops an heuristic for random exploration and two routing strategies in a multi-agent DRL framework for the TUSP. This approach addresses different problem designs, such as different shunting yard types and different matching problems, and to contribute towards a better learning process. The developments help agents make the right decisions from the logistics efficiency point of view and consequently to produce better route plans. The work extends on previous research [2] [26]. This is hence the focus of our research, which constitutes a novel approach from the scientific point of view to implement routing within the TUSP, in a multi-agent DRL framework, especially since the application of artificial intelligence in railway operations are relatively scarce. The main contributions of this research can be summarized as follows:

- An heuristic for random exploration in a multi-agent DRL framework for the TUSP based on the principles of large neighbourhood search.
- Formulation of two routing strategies for the TUSP in a multi-agent DRL framework by means of reward functions.
- Two real-life case studies in the Dutch railway network.
- An analysis of the flexibility of routing strategies to be applied to different types of shunting yards and different matching problems.

The remainder of the paper is structured as follows: Section II provides an overview of related research carried out in the fields of the TUSP and Deep Reinforcement Learning. Section III describes the methodology used in this research. Section IV presents the experiments carried out on two real-life case studies in the Dutch railway network. Finally, Section V presents the conclusions of this research and research directions.

II. Literature Review

In this Section we review relevant research that has been carried out in the TUSP (subsection II-A) including applications with deep reinforcement learning (subsection II-B) and the research gaps identified (subsection II-C).

A. Shunting Yards and the TUSP

In [14], two main types of shunting yards are identified based on the track layout. First, the *carrousel* type, in which tracks are open on both sides and trains move around the shunting yard along different service locations. *Carrousel* shunting yards can either have flow pattern constraints, which means that tracks are in principle used in one single direction, or without flow pattern constraints, which means that tracks can be used in both directions. Second, the shuffleboard type, in which tracks are dead-ended, where trains are stacked following the last-in-first-out (LIFO) principle. Additionally, there are complex layouts that combine the main characteristics of both basic types.

The main subproblems of the TUSP are the following: (1) *Matching:* each train unit is mapped from an incoming unit to an outgoing train unit; (2) *Service:* each train unit has service tasks assigned at service facilities; (3) *Parking:* trains are parked and sorted when they are not being used; (4) *Splitting & combining:* it is necessary to split and combine trains to modify compositions; (5) *Routing:* trains have to be routed from the entry to service locations, parking tracks and to the exit; (6) *Crew scheduling:* crew has to be assigned to the tasks described.

Train departures consist of either lists of requested rolling stock types, without specifying train unit IDs (0% matching type), or lists of requested train unit IDs (100% matching type). Therefore, in the first case, agents have to solve the matching subproblem, whereas in the second case the matching is already given as an input of the problem. Combination is only allowed with trains of the same type. Routing is in essence a rather overarching subproblem since it links the tasks described. As a result, there are clearly strong interactions between the different subproblems and hence the TUSP needs to be approached in an integrated process, which makes it more complex. Several of the TUSP subproblems are already NP-hard [8] [20] [11].

The TUSP has been extensively studied in literature with several approaches using classic operations research techniques [8] [19] [17], which have been shown to be limited, since only some of the TUSP subproblems were formulated (matching, parking and routing), computationally expensive and not scalable to real-life problems. More recently, NS has developed the Stabling Plan Generator (in Dutch: *Opstel Plan Generator*, OPG) [28] based on the work by [17], which included splitting and combination. Also, the TUSP has been approached using a stochastic local search approach [33], which has shown a high performance. Nevertheless, it cannot account for the uncertainty of train arrivals and departures and it is not very consistent in terms of results.

In [6], the author extended on [33] to include the crew scheduling subproblem of the TUSP [6], using the output of the latter as the input. The work by [33] and [6] has been further developed towards the Hybrid Integral Planning tool (HIP) that NS is currently developing [5], which aims at integrating the different subproblems and solving them in parallel. HIP is currently the best performing programme to solve the TUSP.

In [3], heuristics inspired in the container stack assignment and relocation problems have been proposed to improve the occupation rate of shunting yards, since the fact that multiple train units on parking tracks can block each other resembles the physical properties of container stacks. Two heuristics are proposed to park trains: (1) the Type-Based Strategy (TBS), which allocates a single rolling stock type on each track, and (2) the In Residence Time Strategy (IRTS), which sorts trains on each track according to their departure time. A discrete process simulation is used to model the proposed heuristics. The results are compared among them and also with previous work at NS [33] [34] [28] and show that the proposed heuristics have some limitations such as limited flexibility and lower performance compared to more sophisticated mathematical models. Nevertheless, the proposed methods have a high computational efficiency compared to other methods and yield robust results.

B. *Deep Reinforcement Learning Applications in the TUSP*

In the recent years, reinforcement learning (RL) and deep reinforcement learning (DRL) techniques have been applied in the operations research domain to solve the travelling salesman problem [1], the vehicle routing problem [23], resource management problems [21], the multi-driver vehicle dispatching and repositioning problem [12], etc. In recent years, RL has been applied in railway scheduling [15], and DRL in railway rescheduling [24]. Besides, a multi-agent DRL framework for the train rescheduling problem has been researched by the Swiss Federal Railways [27].

Several authors [29] [13] [4] [18] [2] [26] have researched the application of DRL to the TUSP in collaboration with NS, with MDPs and with different agent-based models, machine learning algorithms and state representations.

The first approach to the TUSP with DRL was an event-driven single-agent DRL framework developed to solve the matching, parking and routing subproblems of the TUSP with the possibility of train repositioning [29]. The goal was to explore whether DRL could help to create more consistent route plans that could account for uncertainty in train arrival and departure sequences. The framework uses two MDP formulations, an image-like state representation and Deep Q-Network. The results show that for both formulations (with and without relocation), the agent is able to find consistent strategies with much lower uncertainty than the local search approach [33].

The work by [29] was extended by [13] by incorporating service tasks such as cleaning with different state representations. The model is also event-driven but a waiting action is added to the actions space so that time constraints associated with service tasks are accounted for. Besides, different machine learning algorithms such as Deep Q-network, actor-critic and deep value iteration were tested. The latter method was found to perform better than the rest. Nevertheless, the framework is computationally expensive and, as [29], it does not allow for simultaneous movements.

Successive work moved to a multi-agent system [4]. In this case, the system was composed of two agents, one for the parking tasks and another one for the cleaning tasks. Different approaches were developed, including a basic multi-agent system resembling a single-agent system, and multi-agent systems with full and partial observability. The results revealed better performance of the single-agent system, by contrast to the initial expectations. Also, the coordination between the parking and cleaning agent was found to be problematic in the learning process.

In [18], the author also approached the TUSP with service tasks, although again with a single-agent DRL framework. The state is presented as the visual representation by [29], containing all relevant information of the current timestamp. Policy gradient and Deep Q-network algorithms were tested for agent training. The agent interacts with the environment simulator software built by NS, the Train Maintenance and Shunting Simulator (in Dutch: *Trein Onderhoud en Rangeer Simulator*, TORS) for sequential planning. From the results, it seemed that Q-learning performed better than policy gradient. Nevertheless, the agent was not capable of learning a stable policy for solving scaled-up problems due to a too large action space and on one hand and the state representation used, for which it is difficult to add more functionalities.

The TUSP was approached by [2] using an event-driven multi-agent system in which each train unit is treated as an individual agent, addressing matching, parking and combine and split. The goal was to analyze whether agents can learn how to combine and park properly. By using multiple agents, the action space significantly decreases to only those feasible actions that a particular train unit can perform. A set of positive rewards is given to agents depending on the action taken. In general, the performance of the model measured as the percentage of correct problem instances solved was significantly lower than that of the local search algorithm by [33].

Finally, [26] combines prior work, mostly based on [2], with improvements and extensions with heuristic explorations to stimulate certain actions. Therefore, it features a multi-agent system with one agent per each train unit, MDP formulation and a deep value iteration algorithm.

C. *Research Gap*

Extensive research has been carried out to approach the TUSP using classic operations research techniques (e.g. [8] [19] [17] [33]). These methods fall short in real-life problem cases since the mathematical models are too complex or too restrictive and do not account for uncertainty. Aside, [3] proposed some heuristics to allocate trains to specific tracks to improve shunting movements.

Differently, DRL has been applied to solve train scheduling and rescheduling problems and in the TUSP in particular [29] [13] [4] [18] [2] [26]. Current work in DRL mostly focused

Source	Method	Algorithms	TUSP subproblems	Logistics information	Result evaluation
Peer et al. (2018) [29]	SA DRL	MDP; DQN	M, P, R	No	Benchmarked against [33]
Jamshidi (2019) [13]	SA DRL	MDP; DQN, actor-critic, value iteration	M, S, P, R	No	Compared performance of their own approaches
Cohen (2019) [4]	MA DRL	Markov games; DQN	M, S, P, R	No	Compared performance against SA DRL model
Kyziridis (2019) [18]	SA DRL	MDP; Policy gradient, DQN	M, S, P, R	No	Compared performance of their own approaches
Barnhoorn (2020) [2]	MA DRL	MDP; Value iteration	M, P, R	No	Benchmarked against [33]
NS (2020) [26]	MA DRL	MDP; value iteration	M, S, P, R, S&C	No	-
This paper	MA DRL	MDP; value iteration	M, S, P, R, S&C	Yes	Benchmarked against HIP and [26]

TABLE I
OVERVIEW OF RECENT RESEARCH ON THE TUSP WITH DRL. M: MATCHING; S: SERVICING; P: PARKING; R: ROUTING; S&C: SPLITTING AND COMBINING; SA: SINGLE AGENT; MA: MULTI-AGENT; DQN: DEEP Q-NETWORK.

on the computational aspects, while detailed logistics of real-life problems and have not been approached so far. Following the promising performances of DRL for solving the TUSP, an overview of the related research carried out is shown in Table I, which shows each source, the methodology and algorithms used, the TUSP subproblems tackled, whether they include logistics information and the result evaluation.

The gaps recognised in the current research are as follows. First, the existing research on DRL does not give guidance to agents to park on specific tracks, to perform movements to solve combination and split problems or to reach certain locations within the yard. Consequently, learning is more difficult and hence takes a significantly longer time. Second, they do not account for conflict prevention and resolution, which makes it more difficult for agents to solve instances. Third, they do not prevent unnecessary movements, which we aim at minimizing as much as possible. Fourth, the characteristics of particular shunting yard types are not accounted for and hence it has limited applicability in different yards. Fifth, the efficient use of the parking capacity is not taken into account and therefore it does not maximize capacity. Sixth, there is no distinction between different matching problem types, which again limits its applicability in different problem designs. Lastly, there is a lack of insights on how rewards can be shaped to efficiently solve routing problems.

Consequently, further research is needed to incorporate information from the logistics side of routing in the TUSP to the current framework [26] in order to create routing strategies such that can minimize the number of train movements and hence to help to improve route plans. This is motivated by the work by [3], in which policies from other logistics problems were successfully applied in the TUSP in a different framework. Furthermore, the research on rewards shaping for RL reveal the complexity of designing reward functions such that lead agents to a goal [7], and there are limited insights on how to reduce the search space for agents by means of such functions [22], which play a significant role in speeding up the learning process of agents [10].

Hence, the scientific gap is the development of routing strategies such that take all the above mentioned considerations into account and formulate them in the multi-agent DRL framework [26]. At the higher level, the scientific gap concerns the incorporation of information from the logistics side of routing in the TUSP into a multi-agent DRL framework in order to optimize the output route plans and to contribute towards a better learning process.

This is a novel approach to implement routing within the TUSP, through a multi-agent system and deep reinforcement learning. To the best of the author's knowledge, there are no such approaches in related literature. Besides, the applications of artificial intelligence in the field of railway operations are scarce to date.

III. METHODOLOGY

In this section we present the basic framework used (subsection III-A), the heuristic for random exploration developed (subsection III-B) and the two routing strategies developed (subsection III-C). The methodology is based on a multi-agent system in a deep reinforcement learning framework modelled as a Markov Decision Process (MDP) with value iteration. This research builds on the framework developed by previous work [2] [26].

A. Basic Framework

In [26], authors developed a framework combining two models for representing the TUSP as an MDP. First, TORS, which is the simulation environment for sequential planning, responsible for generating the location and actions. Second, the Multi-Agent Train Unit Deep Reinforcement Learning (MATDRL), which is a programme that uses DRL to solve planning problems for shunting yards. It does so by treating each train unit as a separate agent, and lets the agents choose the actions for their train unit, by which they act upon the environment. After an action is performed, the agent receives a reward and the state is updated. The framework inputs include (1) the set of problem instances (each with arriving and departing trains, arrival and departure times, composition, matching and service tasks scheduled), (2) the location (model of the shunting yard) and (3) the business rules, or problem constraints. Figure 1 shows a high-level representation of the TORS-MATDRL framework for the TUSP.

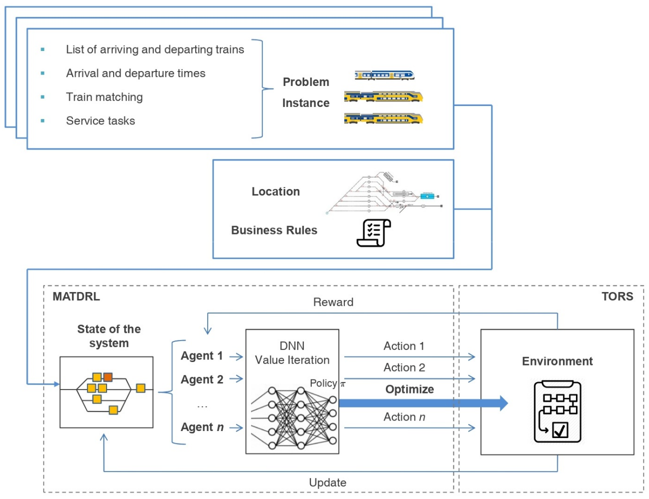

Fig. 1. High-level representation of the multi-agent DRL framework for the TUSP.

The objective of the TUSP in the multi-agent DRL framework is to maximize the number of correct train shunting departures, which is the agents' goal, by maximizing the reward received. By correct departures we mean that each train (1) departs on time from the right track, (2) with the correct composition and (3) with all scheduled service tasks completed, all subject to constraints.

The agents' action space is discrete, $A = \{1, ..., k\}$, and has the following possible actions, which are also represented in Figure 2: (1) *Arrive:* lets a train arrive at the location, reserving all track parts on its route; (2) *Move*: move a train from one track to another specific track; (3) *Set back:* reverses the driving direction of a train; (4) *Combine:* combine two train units of the same rolling stock type into one single train; (5) *Split:* split a train into two train units; (6) *Service:* move and perform a service action on a train unit; (7) *Wait:* let a train wait till the following action occurs. It does not change the current state; (8) *Depart:* let a train leave the shunting yard from the gateway into the passenger service. Each time an action is chosen, the instance advances one step.

The action triggers are the events that prompt agents to act. The following are used: (1) *Arrival:* generated for each arriving train; (2) *Departure:* generated for each time at which a train should depart. Always preceded by *PrepDeparture* to let an agent move to the exit track first. (3) *End Service:* Generated at the end of a service task to allow trains to move towards and away from the service facility. (4) *Middle:* generated in between *Arrival*, *EndService* and *Departure* triggers, depending on the setting of the TORS-MATDRL framework, to allow for combination, splitting, service tasks and relocation. We call relocation to those movements from

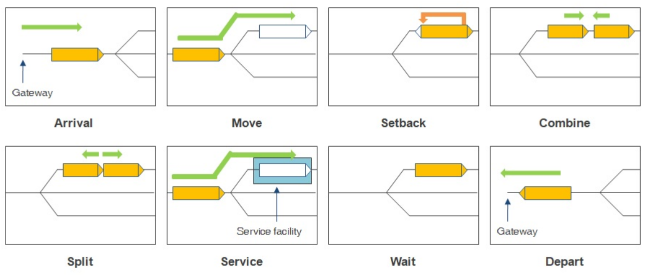

Fig. 2. Actions representation.

a certain parking track to another parallel parking track via a relocation track where the train reverses its direction, and hence involving a total of two movements.

At any action trigger, all invalid actions are masked out so that agents can focus on relevant actions only. Therefore, the distribution and importance of actions in the action space is highly imbalanced, which makes exploration more complex. Also, agents can commit certain violations after which the current episode will end prematurely. An episode is each attempt to solve an instance.

The agent accesses the neural network in order to compute its state values using the Value Iteration algorithm. Agents follow an ε-greedy policy to perform actions, where $0 < \varepsilon < 1$ is a parameter controlling the probability of choosing an action randomly, ε, or a greedy action, $1 - \varepsilon$. Network training starts with $\varepsilon = 1$, which means that agents select possible actions randomly during the exploration phase. During the training process, ε decays gradually, which means that agents will become more likely to select a greedy action.

The environment of an agent comprises non-agent objects, which the agent can observe and act upon. It handles the state representation and the reward function. Since the environment models an MDP, agents only take into account the current state of the system, while previous states are disregarded. The state of the shunting yard is given to the agents in the form of a one-dimensional array. The original reward function (which the routing strategies extend) assigns the following rewards under the following conditions: (1) the action has completed the instance successfully (*Reward.All_left*); (2) the action is *combine* or *split* and the train will match the composition of at least one departing train (*Reward.Comb_correct* and *Reward.Split_correct*, respectively); (3) the action is *service* and service tasks are not completed (*Reward.Start_clean*); (4) the trigger is *End Service*, train corresponds to this trigger and action is *Move* or *Set back* (*Reward.End_clean*); (5) next trigger is *Departure* and action is *move* to gateway track and train is next to depart (*Reward.Move_depart*); (6) current trigger is *Departure* and service tasks completed (*Reward.Leave_clean*).

All agents share the same neural network. The input layer of the neural network is a single array whose size depends on the number of arriving train units in an episode and the network is build using a sequential model with 5 hidden layers, with 512, 256, 128, 64 and 32 and nodes. All nodes use the Rectified

Linear Unit (ReLU) activation functions [9]. The output layer is also fully connected and has one node representing the value of the state. The network is prompted to train on a batch of experiences from its memory, which is called experience replay. The loss function that is used is the Adaptive Moment Estimation optimizer (Adam) [16]. For more details on the network architecture refer to [2].

The agents choose which action to take based on deep value iteration. We define the discount rate γ, ($0 < \gamma < 1$), which is used by the value iteration algorithm and concerns the relative importance of rewards that are received later in the sequence with respect to the rewards obtained at the present moment. Value Iteration creates a policy by computing the value of the policy $V_{n+1}(s)$ at state s by exploring all actions that can be taken from that state onwards and iteratively improving the estimate $V_n(s)$. The algorithm initializes $V_0(s)$ and repeatedly updates the value $V_n(s)$ until a the convergence condition is met [30].

B. Heuristic for Random Exploration

Training in MATDRL starts with agents exploring the search space by choosing actions randomly. Since pure random exploration is computationally very expensive, we develop an Heuristic for Random Exploration (HRE) to efficiently search the state-action space and hence to speed up the learning process. This heuristic stimulates certain actions, based on the principles of large neighbourhood search [31], to help agents identify certain patterns rather than extensive search. This methodology tends to constraint the search space into a local area, which might make the problem more likely to fall into local minima, which is the main limitation of the HRE. Its main advantage is that it significantly helps speeding up the learning process.

The HRE developed is presented in Algorithm 1, where we say that a unit is not in the correct composition when the composition does not match any future departure request. Also, the entry and exit track of the shunting yard is referred to as the gateway track. The HRE gives a weight to each action based on the sum of (1) the weight given by HRE and (2) the actual reward given by the routing strategies (RS, defined in subsection III-C). Agents choose an action based on a weighted random choice. Some examples of action stimulation include setting back instead of waiting, clearing the gateway track, performing a departure or a correct combination.

C. Routing Strategies

In this subsection, the main goal is to develop general routing strategies (RS) that can be applied to different problem designs, including different shunting yard types and matching problems. Therefore, we develop two RS, the Type-Based Routing Strategy (TBS-RS) and the In-Residence Time Routing Strategy (IRTS-RS), which are based on the parking strategies by [3]. Note that parking is only one part of the complete problem and hence we extend these to create complete routing strategies consisting of four components: (1) *standard parking rules*, (2) *combination and split rules*, (3) *conflict resolution*

Algorithm 1 Heuristic for Random Exploration in TORS-MATDRL Pseudocode

```
Input: list of agents and current state
Output: chosen action.
 1: Initialize list of rewards
 2: for All possible actions do
 3:     Compute actual reward (according to RS)
 4:     reward ← actual reward + 0.05
 5:     if action is Move and train is the next to depart then
 6:         if there are still trains to arrive at gateway track then
 7:             if destination track is gateway track and departure is scheduled
                before next arrival then
 8:                 reward ← reward + 0.5
 9:             else if destination track is gateway track and departure is
                scheduled later than next arrival then
10:                 reward ← reward - 0.5
11:             end if
12:         end if
13:     else if action is Wait and train is the next to depart then
14:         if agent is at gateway track and departure is scheduled before next
            arrival then
15:             reward ← reward + 0.25
16:         else if agent is not on gateway track and end time of action
            exceeds time of scheduled departure then
17:             reward ← reward - 0.5
18:         end if
19:     else if action is Wait and train is not the next to depart then
20:         if agent is at gateway track and end time of action exceeds time
            of next scheduled departure then
21:             reward ← reward - 0.5
22:         end if
23:     else if action is Wait and there are still trains to arrive at gateway
        track then
24:         if agent is at gateway track and end time of action exceeds time
            of next scheduled arrival then
25:             reward ← reward - 0.5
26:         end if
27:     else if trigger is PrepDeparture and train is the next to depart and
        action is Move then
28:         if destination track is gateway track then
29:             reward ← reward + 0.75
30:         end if
31:     else if next trigger is Departure and train is the next to depart and
        action is Wait and agent can move in both directions then
32:         if agent is at gateway track then
33:             reward ← reward + 0.75
34:         end if
35:     else if trigger is Departure and train is the next to depart and action
        is Setback then
36:         if agent is at gateway track then
37:             reward ← reward + 0.75
38:         end if
39:     else if action is Depart and train is the next to depart then
40:         reward ← reward + 0.5
41:     else if action is Split or Connect and train is not in the correct
        composition for a departure then
42:         reward ← reward + 0.5
43:     else if action is Setback then
44:         reward ← reward + 0.5
45:     else if action is Wait then
46:         reward ← reward + 0.05
47:     else if action is Service then
48:         reward ← reward + 0.5
49:     end if
50:     if reward <= 0 then
51:         reward = 0.01
52:     end if
53: end for
54: Choose weighted random action (based on rewards).
```

rules and (4) *unnecessary movements rules*. By conflicts, we mean situations that prevent trains from departing on time in the correct composition. The most common example is when trains block each other, but also when two train units to be combined are on different tracks. By unnecessary movements, we mean those movements that are not directly associated with any of the TUSP subproblems and hence could be prevented and still solve the instance correctly. The RS are formulated by means of reward functions. Such rewards assigned to agents create a hierarchy representing the priority of actions over others. We formulate the routing strategies in the TORS-MATDRL framework by means of reward functions. We use the conventions represented in Figure 3.

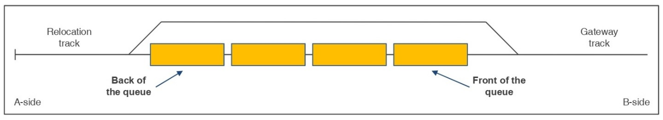

Fig. 3. Representation of the conventions used for routing strategies.

The TBS-RS is based on the rule of grouping trains of the same rolling stock type on the same tracks (Figure 4), such that the four RS components are satisfied. Train units of the same type can consist of a different number of carriages, for which we refer to different subtypes. The TBS-RS assumes no matching is given and hence agents can explore more possibilities for efficient combinations. Any train in the correct composition for the next departure is valid a candidate to perform it. We develop this RS under the assumption that it helps to minimize the number of movements to solve combination problems. The four components of this RS are described as follows: (1) *Standard parking rules:* Movements on parking tracks that result in the arriving train parking directly next to a train unit of the same type are rewarded (*Reward.Park*). If the moving train parks directly next to a train unit of different type, then it will get a penalty (-2**Reward.Park*). If the destination track is empty then it will get no reward. (2) *Combination and Split Rules:* For 0% matching there may be several possibilities for train unit combinations. Rewards are given to combinations of train units that are either a) originally in a composition not matching any future departure (*Reward.Comb_correct*) or b) such that after combining it will match the composition for the next departure (*Reward.Comb_correct*). For all other cases, a penalty is given (-*Reward.Comb_correct*). Splitting is rewarded when the original composition does not match any future departure (*Reward.Split_correct*) and it receives a penalty in any other case (-*Reward.Split_correct*). (3) *Conflict resolution rules:* The preconditions to trigger conflict resolution are I) there are no trains in the front of any queue on any parking track such that are in the composition for the next departure, II) it is neither possible to solve the previous by combining or splitting train units in the front of any queue and III) there are no arrivals left. Two strategies are proposed: a) *Late combination on gateway track:* if there are sufficient candidate train units for the next departure in the front of different parking tracks, then these can move individually to the gateway track (1.5**Reward.Park* for each movement) and combine on it, creating the right composition. *Relocation:* if the previous is not feasible, then if the correct composition is found in the back of any parking track then, it will be prompted to move to the relocation track (2**Reward.Park*), where it will change direction and then to the gateway track via a non-conflicting route (2**Reward.Park*) Figure 5. (4) *Unnecessary movements rules:* If all the previous has not been executed, trains with service tasks completed and standing on a parking track are assigned a penalty if they choose the action *move* (-*Reward.Park*). This also applies at *Departure* for trains that are not next to depart.

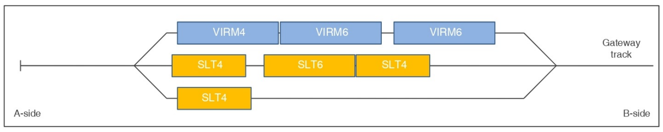

Fig. 4. Example of the TBS standard parking rules with trains of types VIRM and SLT. Each track contains trains of the same rolling stock type.

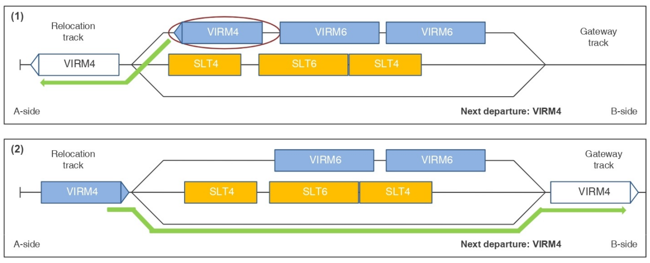

Fig. 5. Representation of the resolution of a conflict by means of relocation. The train unit VIRM4 at the back of the queue is the only possible candidate for the next departure. It moves to the relocation track (1) and then to the gateway track via a non-conflicting path (2).

The In-Residence Time Strategy (IRTS-RS) is based on the rule of assigning train units to parking tracks based on its departure time, creating a chronological order of trains on each track decreasing to the exit side, independently of the rolling stock type (Figure 6), such that the four RS components are satisfied. The IRTS-RS assumes that the matching is given and hence we can park trains according to this rule. The four components of this RS are described as follows: (1) *Standard parking rules:* Movements on parking tracks that satisfy the IRTS rule are rewarded (2**Reward.Park* for movements meeting the train to be combined with according to the matching, *Reward.Park* otherwise). If the rule is violated, or a train unit that has not yet been combined correctly is blocked, a penalty is assigned (-2**Reward.Park*). (2) *Combination and split rules:* With 100% matching the correct combinations or splits are unique. Rewards are assigned to train units for combining or splitting when the train units were not in the correct composition before the operation but correct after the operation (*Reward.Comb_correct* and *Reward.Split_correct*, respectively). (3) *Conflict resolution rules:*

a) *Late combination on gateway track:* If two train units are the next to depart but are still not in the correct composition, and both are in the front of a queue, then we reward them to move individually to the gateway track (2**Reward.Park* for the first unit, *Reward.Park* for the second) and combine on it, creating the right composition. b) *Relocation:* The precondition for a relocation is that the standard parking rules are violated. Relocation consists in moving the train in the back of a queue from the track where the violation has occurred to the relocation track (2**Reward.Park*), where it will change direction, and then to another parking track according to the standard parking rules, until the violation is removed (Figure 7). (4) *Unnecessary movements rules:* If all the previous has not been executed, trains with service tasks completed and standing on a parking track are assigned a penalty if they choose the action *move* (-*Reward.Park*). This also applies at *Departure* for trains that are not next to depart.

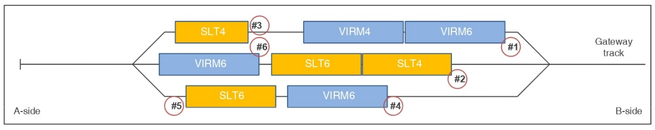

Fig. 6. Example of the IRTS standard parking rules with trains of types VIRM and SLT. The position of each train in the departure sequence is shown next to each train. In this configuration, all trains can depart without blocking each other.

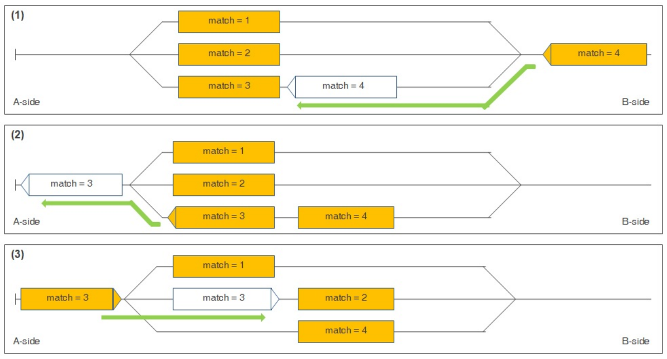

Fig. 7. Representation of a state that creates a conflicting departure sequence (1) and hence it creates the need for relocation, and how relocation can solve the conflict (2-3).

Finally, Table II shows an overview of the main characteristics of the TBS-RS and IRTS-RS in terms of matching subproblem, sources of uncertainty, standard parking rules, combination and split, conflict resolution, unnecessary movements and main application.

IV. Experiments

We use two shunting yards from the Dutch railway network to demonstrate the potential of the framework to produce more efficient route plans and to adapt to different problem designs. First, *carrousel* type without flow pattern constraints, a simplified version of the shunting yard of Heerlen Figure 8, and second, *carrousel* type with flow pattern constraints, the shunting yard of Kleine Binckhorst. An overview of the differences and commonalities between both shunting yards is provided in Table III.

We performed 3 experiments. Firstly, we introduce the experiments setup used (subsection IV-A). Following this, we present Experiment 1 (subsection IV-B), where we assess the performance of both the TBS-RS and the IRTS-RS by benchmarking them against the original settings (i.e. baseline variant) in TORS-MATDRL [26]. By original settings we mean (1) without the routing strategies and (2) with the original reward functions (section III) and (3) we include the heuristic for random exploration. We use a set of instances consisting of 6 train units that have to solve two combinations to form 4 departing trains (Table V). We carry out this experiment in the case study of Heerlen. In Experiment 2 (subsection IV-C) we analyze the performance of the RS to solve the matching and combination subproblems, with 0% and 100% matching. In this case, we incorporate one more train unit and multiple rolling stock types, which have to solve one combination problem in total (Table V). Again we carry out this experiment in the case study of Heerlen. We benchmark the results against HIP. Lastly, in Experiment 3 (subsection IV-D) we extend the previous experiments on matching, combination and servicing to include the splitting subproblem. Instances now consist of a total of 9 train units. We carry out this experiment in the case study of Kleine Binckhorst. We benchmark the results against HIP.

A. Experiments Setup

The two RS developed in subsection III-C are further extended to fit these particular case studies to take into account the location of service facilities. In the case of Heerlen, no flow constraints need to be added, but the service facility serving part of track 33 and 34 has to be incorporated in the RS. Basically, trains will be prompted to clear them as soon as service is completed by moving to the area of the same track without a service facility, or relocate if the preconditions apply. In the case of Kleine Binckhorst, flow constraints are added, by which all trains enter the shunting yard via the entry tracks (56 to 59). The primary flow is depicted in Figure 9, but it is allowed to violate it to let agents park trains on the entry tracks and depart from there under certain preconditions, which replaces the conflict resolution strategies of relocation and combination on the gateway track. The adaptations take into account that all trains with service tasks need to visit the cleaning tracks (61 and 62) before moving to park. Also, combinations are prompted to take place on the storage tracks (52 to 55), whereas splitting is prompted to take place on the relocation track (63). Lastly, moving from the parking tracks to the gateway track too early before the departure time received a significantly higher penalty to prevent trains from closing multiple circles, which is undesired.

Then, we introduce the values for the rewards as shown in Table IV. Also, we generate sets of instances using an Instance Generator [25] such that ensure (1) flow conservation between

	TBS-based Routing Strategy	**IRTS-based Routing Strategy**
Matching Subproblem	Departures defined by train unit types and subtypes required (i.e. not solved)	Departures defined by train unit IDs required (i.e. solved)
Uncertainties	Arriving and departing train units	Arriving train units
Standard Parking	Stacking trains of the same type on the same track	Trains ordered on parking tracks according to departure sequence
Combination and Splitting	Strategy meant to manage numerous possibilities	Unique correct possibilities
Conflict Resolution	(1) Late combination on gateway track (2) Relocation: relocate candidate train or combination on relocation track	(1) Late combination on gateway track (2) Relocation: restoration of chronological parking order
Unnecessary movements	Trains parked, with service tasks completed and no standard parking rules violation wait on its current position until (shortly before) its departure time	
Application	Shunting yard capacity assessment	Planning of daily operations

TABLE II
OVERVIEW OF THE TBS AND IRTS-BASED ROUTING STRATEGIES.

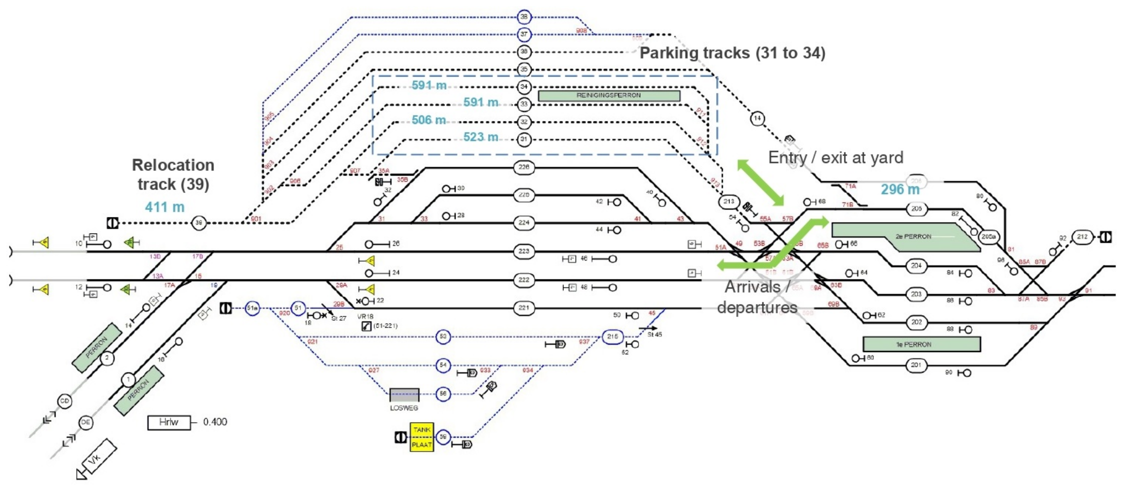

Fig. 8. Heerlen shunting yard (top centre, dashed lines consisting of tracks 31 to 39) and station (right) (adapted from [32]).

	Heerlen	**Kleine Binckhorst**
Type	*Carrousel* shunting yard without flow pattern constraints	*Carrousel* shunting yard with flow pattern constraints
Track usage	Bidirectional	Single direction
Parking tracks	4 free tracks	4 free tracks
Entry tracks	No	4 free tracks
Exit tracks	No	No
Relocation tracks	1 LIFO track	1 LIFO track
Service facilities	Two free tracks directly connected to two parking tracks	Two "drive-through" (free) tracks
Critical tracks	Relocation track	Relocation track

TABLE III
OVERVIEW OF DIFFERENCES AND COMMONALITIES BETWEEN THE SHUNTING YARDS OF HEERLEN AND KLEINE BINCKHORST.

Action	**Description**	**Name**	**Reward**
Service	Start service task	Reward.Start_clean	1.0
Service	Complete service task	Reward.End_clean	0.5
Combine	Combine into correct configuration	Reward.Comb_correct	1.0
Split	Split into correct configuration	Reward.Split_correct	1.0
Move	Correct parking	Reward.Park	0.5
Move	Move towards departure track before departure trigger	Reward.Move_depart	0.75
Depart	Depart with all service tasks completed	Reward.Leave_clean	1.5
Depart	All trains have departed	Reward.All_left	8.0

TABLE IV
LIST OF REWARDS AND THEIR VALUES.

incoming and outgoing train units and that (2) all arrival events occur before all departure events and with sufficient time for

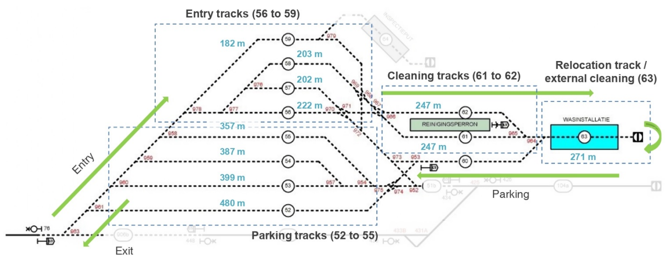

Fig. 9. Kleine Binckhorst shunting yard with operation details (adapted from [32]).

movements and servicing. Arrival and departure sequences of trains are randomized based on given arrival and departure time distributions. We consider rolling stock of types VIRM (Intercity) and SLT (Suburban), operated by NS. Both types can be of subtypes 4 or 6, which indicates the number of carriages in the train unit. Therefore, we refer to them as VIRM4, VIRM6, SLT4 and SLT6. The results obtained with TORS-MATDRL in Experiments 2 and 3 are benchmarked against the results obtained with HIP [5] with a maximum running time of 2 minutes for each instance. Lastly, the *Middle* action triggers to be added are defined for each particular problem to give agents sufficient chances to perform actions.

Besides, we use a set of performance metrics to evaluate (1) the parking events, (2) combination and splits, (3) movements and (4) the parking capacity consumption. We define the lower bound of the number of combinations as the minimum number of combinations needed to solve a particular instance given the list of arriving and departing trains. Therefore, the expected number of combinations normalized on the lower bound is always 1.0. If a particular solution to an instance performs more combinations than what is strictly necessary, then this value will be greater than 1.0. Besides, to measure the parking capacity, we calculate first the consumption of parking tracks as the maximum number of tracks occupied simultaneously out of the total amount of parking tracks considered, and second, the consumption of parking capacity as the total track length occupied out of the total available track length of the parking tracks used. The performance metrics are only evaluated for correctly solved instances.

The following learning parameters are used in TORS-MATDRL. The learning rate α is set to 0.00002 and the discount factor γ is set to 0.9999, which makes agents take highly into account future rewards. The network memory is capped at 125,000 steps, from which 256 steps are sampled every time the network trains. The target network synchronizes with the weights of the main network once every 200 episodes. Experience replay gives more importance to recent experiences. The length of the exploration phase is measured in steps and it is tuned differently for each problem, as well as the ε decay. For Experiment 1, the exploration phase has 50,000 steps and the ε decay equals 1 / 10,000. In Experiment 2, the exploration phase has 50,000 steps and the ε decay equals 1 / 15,000. In Experiment 3, the exploration phase has 80,000 steps and the ε decay equals 1 / 12,000. The reason for a longer exploration is that instances in Kleine Binckhorst involve more movements than instances in Heerlen and hence more steps are needed to be explored sufficiently.

Table V shows the lists of arriving and departing trains on each experiment and Table VI shows the *Middle* triggers added on each experiment. 150 instances are generated for training in each experiment as well as a testing set for each. The experiments are carried out using an Intel Core i7-7500U CPU with 16 GB RAM.

B. *Experiment 1*

The computational results are shown in Table VII, which presents the number of episodes performed during training, the total training time, the maximum solvability achieved and the value of ε when this was achieved. The performance metrics are shown in Table VIII. The results in Table VII show a very low solvability for the models with no RS (below 10%), which have not converged after long training time, and is significantly improved when the RS are included, up to 56% in the TBS-RS. The reason is that with no RS, agents find it difficult to learn events involving movements such as correct parking or correct combinations. When comparing the TBS-RS with the IRTS-RS, we observe a solvability of up to 56.0% after about 27 hours of training in the former and up to 44.5% after about 14 hours of training. The training process shows a slower convergence for the TBS-RS compared to the IRTS-RS, as a result of the former being a more relaxed problem in which agents have to explore the state-action space more extensively to solve the matching problem. The increased number of possibilities to solve the problem also explains a higher solvability achieved with 0% matching. With 100% matching, by contrast, the problem is more constrained and hence the algorithm seems to converge faster. Regarding instances not solved, the most common violations concern trains being unable to depart, which means that at the time of its scheduled departure, the corresponding train was not on the departure track. This usually happens when there are conflicts, when despite the conflict resolution strategies, the policy learned with training does not match perfectly the expected behaviour, and other agents with higher expected q-values for certain other actions may act instead in a given action trigger. As a result, some of the actions of the required sequence of actions may be delayed in time to subsequent action triggers, and sometimes too late.

In terms of performance metrics (Table VIII), the RS decrease the total number of movements by 25% and increase the duration of parking events on parking tracks by about 80-120%, while decreasing the occupancy of the relocation track from 40-50% to 3-13%, which is better since we want to clear it as soon as possible for other trains to use it for conflict resolution. Also, the TBS-RS decreases the average number of combinations from 5.38 to 1.05, which gets very close to the lower bound. Besides, the RS increase the average number of parking events according to the corresponding parking rule, which means that trains tend more to park on

	Experiment 1		Experiment 2		Experiment 3	
	Arriving trains	**Departing trains**	**Arriving trains**	**Departing trains**	**Arriving trains**	**Departing trains**
1	[SLT4]	[SLT6]	[SLT6]	[SLT6]	[SLT4]	[SLT4]
2	[SLT4]	[SLT6]	[SLT6]	[SLT6]	[SLT6]	[SLT6]
3	[SLT6]	[SLT4, SLT6]	[SLT6]	[SLT6]	[SLT6]	[SLT6]
4	[SLT6]	[SLT4, SLT6]	[VIRM4]	[VIRM4]	[VIRM4]	[VIRM4]
5	[SLT6]		[VIRM4]	[VIRM4, VIRM6]	[VIRM4]	[VIRM4, VIRM4]
6	[SLT6]		[VIRM6]	[VIRM6]	[VIRM4, VIRM6]	[VIRM6]
7			[VIRM6]		[VIRM6]	[VIRM6]
8					[VIRM6]	[VIRM6]

TABLE V
LIST OF ARRIVING AND DEPARTING TRAINS PER EXPERIMENT (UNORDERED).

	Experiment 1		Experiment 2		Experiment 3	
	TBS-RS 0% matching	**IRTS-RS 100% matching**	**TBS-RS 0% matching**	**IRTS-RS 100% matching**	**TBS-RS 0% matching**	**IRTS-RS 100% matching**
Rank	Triggers	Triggers	Triggers	Triggers	Triggers	Triggers
$i-1$		Middle		Middle		
i	*Arrival*	*Arrival*	*Arrival*	*Arrival*	*Arrival*	*Arrival*
$i+1$	Middle	Middle	Middle	Middle	Middle	Middle
j	*End Service*	*End Service*	*End Service*	*End Service*	*End Service*	*End Service*
$j+1$					Middle	Middle
$j+2$					Middle	Middle
$k-2$	Middle		Middle	Middle	Middle	
$k-1$	Middle	Middle	Middle	Middle	Middle	Middle
k	*Prep Departure*	*Prep Departure*	*Prep Departure*	*Prep Departure*	*Prep Departure*	*Prep Departure*

TABLE VI
MIDDLE TRIGGERS ADDED PER EXPERIMENT.

partially filled tracks. When comparing the TBS-RS and IRTS-RS it is noticeable that the number of correct parking events according to the applying parking rule is significantly higher in the TBS-RS, with 5.13, compared to the IRTS-RS, with 3.23. The reason is that the probability of finding a train unit of the same rolling stock type (TBS rule) is higher than finding a train unit that is scheduled to depart later (IRTS rule). Therefore, with the IRTS parking rule it is logically more likely for agents to park on empty tracks. This observation goes in accordance with the fact that the TBS-RS tends to occupy fewer tracks simultaneously (70%) than the IRTS-RS (80%), whereas the tracks being used are occupied up to 45.4% and 34.8% of its length, respectively. This suggests that the TBS-RS makes more efficient use of the parking capacity compared to the IRTS-RS. Lastly, the average number of train movements per train is around 2.4-2.6 for both RS.

Figure 10 presents visualizes the solution produced by TORS-MATDRL for one particular instance with IRTS-RS. The figure represents the task schedules for each train unit in the problem, indicated by their corresponding ID on the vertical axis. Time is represented on the horizontal axis whereas each row represents one train unit. In this particular instance, train units 2601 and 2402 combine to form one of the departing trains, as well as 2602 and 2401. Therefore, all train units within the same train perform the same actions at the same times.

In conclusion, the results of the experiment show that the implementation of both the TBS-RS and the IRTS-RS in TORS-MATDRL has very positive effects on the overall learning process and in the quality of the produced route plans.

C. *Experiment 2*

The computational results are shown in Table VII and the performance metrics in Table IX. The computational results show a lower solvability compared to the previous experiment, now around 30-35%, after adding one more train unit and multiple rolling stock types. Again, the convergence of the algorithm seems to be faster with the IRTS-RS due to the more constrained problem compared to the TBS-RS. In terms of performance metrics (Table IX), the number of correct IRTS parking events in the IRTS-RS has increased from 3.0 to 4.2 due to the presence of more trains, whereas the number of correct TBS parking events in the TBS-RS has decreased from 5.1 to 4.0 due to the presence of multiple rolling stock types. With regards to the parking capacity consumption, we observe again that the TBS-RS uses the parking capacity on average more efficiently than the IRTS-RS, as the TBS-RS occupies 70.8% of the tracks at 57.0% of its length, compared to 80.0% and 51.8% respectively in the IRTS-RS. When comparing with HIP, the main observation is that HIP often lets trains standstill on the entry track, which do not enter the shunting yard and hence the total number of movements achieved is lower than those obtained in TORS-MATDRL (about 12.5 compared to 16.5-18.0). This does not affect the feasibility of the produced route plans but it is undesired, By contrast, TORS-MATDRL with the RS generally makes all trains enter the shunting yard. Therefore, it is difficult to retrieve solid conclusions on how movements are handled in both frameworks. Nevertheless, some other things can be noted. In the problem with 0% matching, HIP succeeds in meeting the exact lower bound of the number of combinations required while TORS-MATDRL does not, albeit it gets very close to it (1.17 compared to

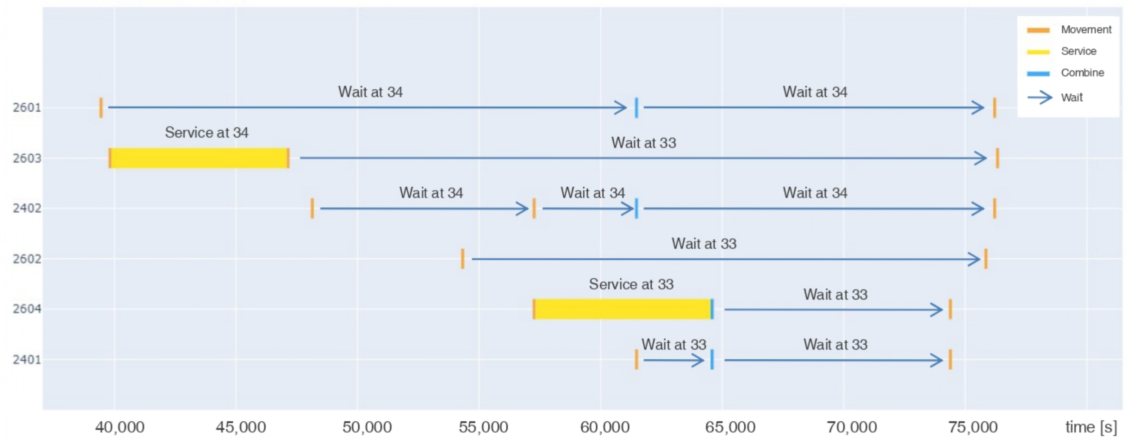

Fig. 10. Task schedule representing the solution to one particular instance in Heerlen by TORS-MATDRL with IRTS-RS.

	Experiment 1				Experiment 2		Experiment 3	
Training	**0% matching**		**100% matching**		**0% matching**	**100% matching**	**0% matching**	**100% matching**
	No RS	**TBS-RS**	**No RS**	**IRTS-RS**	**TBS-RS**	**IRTS-RS**	**TBS-RS**	**IRTS-RS**
Number of episodes	5,000	5,200	4,400	3,400	4,600	4,400	4,400	4,800
Training time [h]	23.0	26.8	16.1	13.8	34.3	40.9	74.9	69.9
Maximum solvability (200 episodes)	7.5%	56.0%	3.0%	44.5%	32.5%	31.5%	15.0%	14.5%
End ε	0.6032	0.7546	0.6645	0.7924	0.9037	0.9697	0.7767	0.8194

TABLE VII
COMPUTATIONAL RESULTS OF THE TRAINING PROCESS PER EXPERIMENT.

		0% matching		100% matching	
Category	**Performance Indicator**	**No RS**	**TBS-RS**	**No RS**	**IRTS-RS**
Standard parking	Average number of correct TBS parking events	3.83	5.13	3.54	3.23
	Average number of correct IRTS parking events	-	-	2.36	3.00
	Average number of empty parking events	3.00	2.69	3.45	3.32
Combination and split	Average number of combinations normalized on lower bound	5.38	1.05	1.00	1.00
	Average number of splits	8.75	0.09	0.00	0.00
	Average number of combinations on gateway track	5.50	0.13	1.45	0.45
Movements	Average total number of movements	15.50	14.74	16.80	12.68
	Average number of relocation movements	1.58	0.95	1.09	0.32
Capacity consumption	Average duration of parking events on parking tracks [s]	7,765.9	13,818.8	7,046.0	15,591.4
	Average duration of parking events on relocation track [s]	9,967.2	3,147.7	9,485.3	4,127.6
	Occupancy of relocation track [%]	52.3%	13.2%	44.4%	3.4%
	Average consumption of parking tracks [%]	58.3%	70.5%	84.1%	79.5%
	Average consumption of parking capacity [%]	38.9%	45.4%	36.5%	34.8%

TABLE VIII
PERFORMANCE METRICS FOR EXPERIMENT 1.

the lower bound of 1.00). Also, TORS-MATDRL seems to perform significantly better in terms of occupying critical tracks such as the relocation track as little as possible (less than 10% of the time) compared to HIP (more than 50% of the time).

In conclusion, in this experiment we have shown the potential of the TBS-RS and the IRTS-RS to solve matching, combination and servicing problems with multiple rolling stock type in a *carrousel* shunting yards without flow pattern constraints.

D. Experiment 3

The computational results are shown in Table VII and the performance metrics in Table X. The results show a significantly lower solvability compared to the previous experiments, now reaching up to 15% after a significantly longer training time (up to 3 days compared to less than 2 days in the previous). The most common violations are associated with trains being unable to leave due to a bottleneck in the relocation track, which most train use. Despite the rewards assigned, the policy learned during training does not match perfectly the expected behaviour agents of clearing the relocation track as soon as possible, and other agents with higher expected q-values for certain other actions may act instead in a given action trigger. In terms of performance metrics, we observe again TBS parking events (with TBS-RS) happening more often than IRTS parking events (with IRTS-RS), with 5.14 and 2.60 events on average respectively. The average number of

		0% matching		100% matching	
Category	**Performance Indicator**	**TORS-MATDRL TBS-RS**	**HIP**	**TORS-MATDRL IRTS-RS**	**HIP**
Standard parking	Average number of correct TBS parking events	4.0	1.4	3.1	1.3
	Average number of correct IRTS parking events	-	-	4.2	1.2
	Average number of empty parking events	2.9	3.3	3.5	3.6
Combination and split	Average number of combinations normalized on lower bound	1.17	1.00	1.00	1.00
	Average number of splits	0.17	0.00	0.00	0.00
	Average number of combinations on gateway track	0.0	0.1	0.1	0.0
Movements	Average total number of movements	16.5	12.3	18.2	12.7
	Average number of relocation movements	0.4	0.5	1.2	0.4
Capacity consumption	Average duration of parking events on parking tracks [s]	16,260.4	20,400.1	17,059.7	20,116.1
	Average duration of parking events on relocation track [s]	181.0	17,328.3	3,467.3	19,882.6
	Occupancy of relocation track [%]	0.3%	83.8%	9.3%	93.5%
	Average consumption of parking tracks [%]	70.8%	81.3%	80.0%	90.0%
	Average consumption of parking capacity [%]	57.0%	34.4%	51.8%	32.7%

TABLE IX
PERFORMANCE METRICS FOR EXPERIMENT 2.

parking events on empty tracks is close to 4 in both RS, which means that all parking tracks 52 to 55 (4 tracks in total) will be used in most cases. With regards to combination and split, both RS perform the exact lower bound of combinations and splits. Combinations take place on parking tracks and splits on the relocation track as designed. Regarding movements, we can assume that each train needs at least 4 movements to close the entire *carrousel* (arrival at the facility, move to relocation track, then to parking track, and finally depart), whereas the RS produce an average of 4.4-4.9 movements per train, which suggests that the RS produce some possibly unnecessary movements. Regarding the parking capacity consumption, the results suggest that with higher congestion in this case study, compared to the case studies in the previous experiments, the differences in how the parking capacity is consumed are reduced. In fact, on average, all storage tracks are used (90-100%) in both cases, at about 70% of its track length.

When comparing with HIP, we observe fundamental differences in how movements are handled. While the RS starts filling in the storage tracks first as much as possible before starting using the entry tracks for parking, HIP considers parking on the entry tracks much more flexibly and hence trains are parked much more spread throughout the shunting yard rather than stacking them as much as possible as they are arriving. This is indicated by the exits classification by location. TORS-MATDRL makes more than 90% of the trains depart from the parking tracks, whereas HIP only dispatches less than 30% of the trains from those tracks, whereas the rest are dispatched from either the entry tracks or facility tracks. As a result, the average number of movements with HIP is lower (about 23-24 compared to 35-39) since departing from the entry tracks (and hence not closing the *carrousel*) involves fewer movements than actually closing the *carrousel*, which most trains with the RS do. Therefore, in conclusion, the RS do not seem to be the most efficient strategies to minimize the number of movements in problems with 9 train units, since HIP finds a different strategy involving fewer movements. We cannot guarantee however that this will also hold for scaled-up problems, which possibly will not since a more congested infrastructure may lead to a different way of moving trains in HIP to better exploit parking capacity. The logic behind the RS implemented in Kleine Binckhorst already tries to maximize the parking capacity rather than the number of movements.

In conclusion, this experiment has shown the potential of the TBS-RS and the IRTS-RS to solve matching, combination, splitting and servicing problems in a *carrousel* shunting yard with flow pattern constraints. The problem is computationally more complex than the previous experiments and the intensive use of the relocation track creates a remarkable bottleneck that agents with the RS find it difficult to prevent.

V. CONCLUSIONS

In this paper, we have developed an heuristic for random exploration and two routing strategies for the Train Unit Shunting Problem in a multi-agent deep reinforcement learning framework adapted for two different case studies. The application of the implementations in two real-life case studies in the Dutch railway network has demonstrated its potential to produce more efficient route plans and to adapt to different problems designs such as different matching problems types and different shunting yards. For problems with 6 train units, the routing strategies have increased the solvability from less than 10% to up to 55% and reduce the total number of movements by 25%. The average number of movements per train in the case study of Heerlen is about 2.1-2.6, whereas in Kleine Binckhorst it ranges between 4.4 and 4.9. Problems with 100% matching seem to converge faster than problems with 0% matching, which are more relaxed. Shunting yards with flow pattern constraints also seem to be more constrained than those without such constraints, and agents tend to create a bottleneck on the relocation track, which they find difficult to resolve. The solvability achieved in this case study is up to 15% compared to 30-55% in the experiments case study without flow constraints.

The TORS-MATDRL framework, with the extensions developed in this research, addresses a real-life problem and the solutions produced by this framework have practical effects on problems such as real-time train shunting due to the instant

		0% matching		**100% matching**	
Category	**Performance Indicator**	**TORS-MATDRL TBS-RS**	**HIP**	**TORS-MATDRL IRTS-RS**	**HIP**
Standard parking	Average number of correct TBS parking events	5.14	1.79	3.33	1.73
	Average number of correct IRTS parking events	-	-	2.60	1.53
	Average number of empty parking events	3.93	1.00	3.80	0.66
Combination and split	Average number of combinations normalized on lower bound	1.00	1.00	1.00	1.00
	Average number of splits	1.0	1.00	1.00	1.00
Movements	Average total number of movements	35.4	23.4	39.3	24.1
	Average number of reversal movements	0.0	2.2	2.8	2.9
	Exits from parking tracks [%]	92.3%	27.6%	91.9%	27.7%
	Exits from rest of tracks [%]	6.8%	53.7%	5.7%	49.7%
Capacity consumption	Average duration of parking events on parking tracks [s]	19,331.0	23,234.7	18,260.6	19,621.6
	Average duration of parking events on relocation track [s]	2,688.2	9,772.3	1,690.2	11,916.4
	Occupancy of relocation track [%]	55.7%	62.6%	36.8%	75.2%
	Average consumption of parking tracks [%]	94.6%	46.4%	91.6%	38.3%
	Average consumption of parking capacity [%]	70.5%	71.1%	70.9%	74.4%

TABLE X
PERFORMANCE METRICS FOR EXPERIMENT 3.

inference of results after training. Our approach can be applied to a broader range of shunting yards types for any mainline railway, metro or tram network by adjusting the components of the RS to fit particular problem designs. Even broader, we believe that similar approaches with collaborative multi-agent cooperation tasks can be developed to solve other routing problems in the field of transportation and logistics. Our results demonstrate that DRL has the potential to solve routing optimization problems efficiently and effectively. Nevertheless, more research is needed before artificial intelligence can represent a solid alternative to classic operations research approaches to certain problems.

Possible further research directions have been identified, such as exploring the impact of learning parameters on performance, further investigating the behaviour of agents to better tackle conflicts and bottlenecks and the effects of reward shaping to optimize learning. We also suggest exploring variations of the methodology used in this research by enhancing learning with heuristics or combining it with optimization approaches. Besides, we advise including new constraints such as crew scheduling. Finally, we recommend scaling up the input instances to test the scalability of the developed heuristics and to eventually solve real-life instances.

ACKNOWLEDGEMENTS

We would like to thank Bob Huisman, Demian de Ruijter, Pim van den Bogaerdt, Joris den Ouden and Roel van den Broek from the Research & Development Node Logistics department at the Dutch Railways NS for their invaluable support to make this research possible.

REFERENCES

[1] MM. Alipour and SN. Razavi. A new multiagent reinforcement learning algorithmto solve the symmetric traveling salesman problem. *Multiagent and Grid Systems*, 11(2):107–119, 2015.

[2] Q. Barnhoorn. A Multi-Agent Approach to the Train Unit Shunting Problem. Bachelor's Thesis in Artificial Intelligence. Radboud University, 2020.

[3] E. Beerthuizen. Optimizing train parking and shunting at NS service yards. Master's Thesis, TU Delft, Delft, 2018. URL http://resolver.tudelft.nl/uuid: d5c93539-f182-4d4d-931b-c5356727c827.

[4] I. Cohen. A multi-agent deep reinforcement learning approach to the train unit shunting problem. Jheronimus Academy of Data Science (JADS), 2019.

[5] J. Den Ouden. The inner workings of HIP. Technical report, NS, Utrecht, 2019.

[6] J. Den Ouden. Generating robust schedules for train maintenance staff. Master's thesis, Utrecht University, 2018. URL http://dspace.library.uu.nl/bitstream/handle/1874/362947/thesis.pdf?sequence=2&isAllowed=y.

[7] D. Dewey. Reinforcement learning and the reward engineering principle. In 2014 AAAI Spring Symposium Series, 2014.

[8] R. Freling, RM. Lentink, LG. Kroon, and D. Huisman. Shunting of passenger train units in a railway station. Transportation Science, 39(2):261–272, 2005.

[9] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In Proceedings of the 14th International Conference on *Artificial Intelligence and Statistics (AISTATS)*, pages 315–323, 2011.

[10] P. Goyal, S. Niekum, and RJ.Mooney. Using natural language for reward shaping in reinforcement learning. 2019.

[11] JT. Haahr, RM. Lusby, and JC. Wagenaar. Optimization Methods for the Train Unit Shunting Problem. European Journal of Operational Research 262, 262(3):981–995, 2017.

[12] J.Holler, R. Vuorio, Z. Qin, X. Tang, Y. Jiao, T. Jin, S. Singh, C.Wang, and J. Ye. Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. In 2019 IEEE International Conference on Data Mining (ICDM), 2019. doi: 10.1109/ICDM.2019.00129.

[13] H. Jamshidi. Application of Deep Learning in Train Shunting Problem. Technical Report, Delft University of Technology and NS, 2019.

[14] S.E.B. Janssens. Empirical Analysis of Service Locations at NedTrain. A closer look at the stabling and handling capacity. Master's Thesis, TU Delft, Delft, 2017. URL https://repository.tudelft.nl/islandora/object/uuid%3Af50c79b5-ee5b-4016-b9db-f075dd8bf417.

[15] H. Khadilkar. A scalable reinforcement learning algorithm for scheduling railway lines. *IEEE Transactions on Intelligent Transportation Systems*, 20(2):727–736, 2018.

[16] DP. Kingma and JL. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations ICLR*, 2015.

[17] LG. Kroon, RM. Lentink, and A. Schrijver. Shunting of passenger train units: An integrated approach. Transportation Science, 42(4):436–449, 2008.

[18] G. Kyziridis. Deep Reinforcement Learning Algorithms for the Train Unit Shunting Problem. Master's Thesis in Computer Science. Leiden University, 2019.

[19] RM. Lentink, PJ. Fioole, LG. Kroon, and C. Van't Woudt. Applying Operations Research techniques to planning of train shunting. Technical report, Erasmus University, Rotterdam, 2003.

[20] RM. Lentink. Algorithmic Decision Support for Shunt Planning. Technical report, Erasmus University Rotterdam, 2006.

[21] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56, 2016.

[22] L. Matignon, GJ. Laurent, and N. Le Fort-Piat. Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning. In International Conference on Artificial Neural Networks, pages 840–849. Springer, 2006.

[23] M. Nazari, A. Oroojlooy, L. Snyder, andM. Takác. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pages 9839–9849, 2018.

[24] L. Ning, Y. Li, M. Zhou, H. Song, and H. Dong. A deep reinforcement learning approach to high-speed train timetable rescheduling under disturbances. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3469–3474, 2019.

[25] NS. Instance Generator. Technical report, NS Knooppunt Logistiek, 2020.

[26] NS. TORS and MATDRL, 2020.

[27] S. Mohanty, E. Nygren, F. Laurent, M. Schneider, C. Scheller, N. Bhattacharya, J. Watson, A. Egli, C. Eichenberger, C. Baumberger, G. Vienken, I. Sturm, G. Sartoretti, and G. Spigler. Flatland-rl: Multiagent reinforcement learning on trains, 2020. URL https://arxiv.org/abs/2012.05893

[28] NS. Functionele beschrijving Opstel Plan Generator (OPG). Technical report, NS, 2018. URL www.ortec.com

[29] E. Peer, V. Menkovski, Y. Zhang, and WJ. Lee. Shunting trains with deep reinforcement learning. *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2018. doi: 10.1109/SMC.2018.00520

[30] WB. Powell. Reinforcement Learning and Stochastic Optimization. A Unified Framework for Sequential Decisions. John Wiley Sons, Inc, 2020.

[31] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In Principles and Practice of Constraint Programming - CP982014, pages 417–431. Springer, 1998.

[32] Sporenplan Online. Sporenplan Online, 2019. URL http://www.sporenplan.nl.

[33] RW. Van Den Broek. Train Shunting and Service Scheduling: an Integrated Local Search Approach. Master's Thesis, Utrecht University, Utrecht, 2016.

[34] F. Wolfhagen. The train unit shunting problem with reallocation. Master's thesis, Erasmus University Rotterdam, 2017. URL https://thesis.eur.nl/pub/37696.

B Reward Functions

Algorithm 3 shows the formulation in the TORS-MATDRL framework of the procedure for the TBS-based routing strategy in Heerlen (two parts).

Algorithm 4 shows the formulation in the TORS-MATDRL framework of the procedure for the IRTS-based routing strategy in Heerlen (three parts).

Algorithm 5 shows the formulation in the TORS-MATDRL framework of the procedure for the TBS-based routing strategy in Kleine Binckhorst (two parts).

Algorithm 6 shows the formulation in the TORS-MATDRL framework of the procedure for the IRTS-based routing strategy in Kleine Binckhorst (two parts).

Algorithm 3 Heuristics for Assignment of Rewards with TBS-Based Routing Strategy in Heerlen Pseudocode in TORS-MATDRL (Part 1)

```
    Input: train unit $u \in U$, action chosen, event trigger, list of agents.
    Output: reward.
 1: if the action completed the instance successfully then
 2:     return Reward.All_left
 3: else if event is Middle and action is Move then
 4:     if origin track $\notin T_p$ and destination track $\in T_p$ then
 5:         List trains parked on destination track $\tau$
 6:         if there is at least one unit parked on $\tau \in T_p$ and current trigger is at least 3 triggers preceding next Departure then
 7:             if train arrives from A-side then
 8:                 if train parked in the front of the queue is of the same type as arriving unit then
 9:                     return Reward.Park
10:                 else
11:                     return -2*Reward.Park
12:                 end if
13:             end if
14:         else if origin track $\in T_r$ then
15:             if there are no candidates in queues for next departure and no Arrivals left then
16:                 if unit is in the correct composition for the next departure and destination track is an empty track $\tau \in T_p$ then
17:                     return 2*Reward.Park
18:                 end if
19:             end if
20:         else if origin track $\in T_p$ then
21:             List trains parked on origin track $\tau$
22:             if gateway track $\in T_g$ is empty and preconditions for conflict resolution are met then
23:                 if unit and another unit on a different track $\in T_p$ are candidate to combine and match composition for next departure
    and destination track $\in T_g$ then
24:                     return 1.5*Reward.Park
25:                 else if unit is in the correct composition for next departure and unit is blocked on gateway side and destination track
    $\in T_r$ is empty then
26:                     return 1.5*Reward.Park
27:                 else if there is no correct composition for next departure and unit is part of the correct composition and unit is blocked
    on gateway side and destination track $\in T_r$ is empty then
28:                     return 1.5*Reward.Park
29:                 end if
30:             end if
31:             if unit is parked in the front of the queue and is to be combined and with unit on gateway track $\in T_g$ to match composition
    for next departure and destination track $\in T_g$ then
32:                 return Reward.Park
33:             else if unit is in correct composition and service tasks completed and origin track $\in T_p$ then
34:                 return -2*Reward.Park
35:             else if unit is not in correct composition and service tasks completed and origin track $\in T_p$ then
36:                 if train to be combined with is being serviced or has not arrived yet at the shunting yard then
37:                     return -Reward.Park
38:                 end if
39:             end if
40:         end if
41:     end if
42: else if action is Connect then
43:     if unit is not in a correct composition or it will create the composition for the next departure after the operation then
44:         return Reward.Comb_correct
45:     else
46:         return -Reward.Comb_correct
47:     end if
48: end if
```

Algorithm 3 Heuristics for Assignment of Rewards with TBS-Based Routing Strategy in Heerlen Pseudocode in TORS-MATDRL (Part 2)

```
1: if action is Split then
2:     if unit is not in the correct composition and is parked on $\tau \in T_p$ then
3:         return Reward.Split_correct
4:     else if unit is in the correct composition for next departure then
5:         return -Reward.Split_correct
6:     else if at least one of the involved units will be in incorrect composition after the operation then
7:         return -Reward.Split_correct
8:     end if
9: else if action is Service and assigned service tasks are not completed then
10:     return Reward.Start_clean
11: else if event is EndService and unit corresponds to the event and action is SetBack then
12:     Reward.End_clean
13: else if event is EndService and action is Move then
14:     if unit corresponds to the event then
15:         if destination track $\in T_g$ or $\in T_r$ then
16:             return -Reward.Park
17:         else if destination track $\in T_p$ then
18:             return Reward.End_clean
19:         end if
20:     else if unit does not correspond to the event then
21:         if origin track $\notin T_p$ and destination track $\in T_p$ then
22:             List trains parked on destination track $\tau \in T_p$
23:             if there is at least one unit parked on $\tau \in T_p$ then
24:                 if train arrives from A-side then
25:                     if train parked in the front of the queue is of the same type as arriving unit then
26:                         return Reward.Park
27:                     else
28:                         return -2*Reward.Park
29:                     end if
30:                 end if
31:             end if
32:         else if origin track $\in T_p$ then
33:             if unit is in correct composition and service tasks completed and origin track $\in T_p$ then
34:                 return -2*Reward.Park
35:             else if unit is not in correct composition and service tasks completed and origin track $\in T_p$ then
36:                 if train to be combined with is being serviced or has not arrived yet at the shunting yard then
37:                     return -Reward.Park
38:                 end if
39:             end if
40:         end if
41:     end if
42: else if event is PrepDeparture and action is Move and train is in correct composition for next departure then
43:     if destination track $\in T_g$ then
44:         return Reward.Move_depart
45:     end if
46: else if action is Depart and service tasks are completed then
47:     return Reward.Leave_clean
48: else if action is Depart then
49:     return Reward.Leave_dirty
50: else if current event or next event is Departure and action is Move then
51:     if service tasks are completed and origin track $\notin T_g, T_r$ then
52:         return -Reward.Park
53:     end if
54: end if
55: return 0
```

Algorithm 4 Heuristics for Assignment of Rewards with IRTS-Based Routing Strategy in Heerlen Pseudocode in TORS-MATDRL (Part 1)

```
    Input: train unit $u \in U$, action chosen, event trigger, list of agents.
    Output: reward.
 1: if the action completed the instance successfully then
 2:     return Reward.All_left
 3: else if action is Connect and unit is not in the correct composition and connection will result in the correct composition then
 4:     return Reward.Comb_correct
 5: else if action is Split and unit is not in the correct composition then
 6:     return Reward.Split_correct
 7: else if event is Middle and action is Move then
 8:     if origin track $\notin T_p$ and destination track $\in T_p$ then
 9:         List trains parked on destination track $\tau$
10:         if there is at least one unit parked on $\tau \in T_p$ then
11:             if train arrives from A-side then
12:                 if train parked in the back of the queue is to be combined with arriving train then
13:                     return 2*Reward.Park
14:                 else if train parked in the back of the queue is to be combined but not with arriving train then
15:                     return -2*Reward.Park
16:                 else if arriving train departs later than train parked in the back of the queue then
17:                     return Reward.Park
18:                 else if arriving train departs earlier than train parked in the back of the queue then
19:                     return -2*Reward.Park
20:                 end if
21:             else if train arrives from B-side then
22:                 if train parked in the front of the queue is to be combined with arriving train then
23:                     return 2*Reward.Park
24:                 else if train parked in the front of queue is to be combined but not with arriving train then
25:                     return -2*Reward.Park
26:                 else if arriving train departs earlier than train parked in the front of the queue then
27:                     return Reward.Park
28:                 else if arriving train departs later than train parked in the front of the queue then
29:                     return -2*Reward.Park
30:                 end if
31:             end if
32:         end if
33:     else if origin track $\in T_p$ then
34:         List trains parked on origin track $\tau$
35:         if train is parked in the front of the queue and is to be combined and is next to depart and destination track $\in T_g$ and 3 triggers
    before its scheduled departure then
36:             return Reward.Park
37:         else if the queue on $\tau \in T_p$ consists of more than one unit and parking order is violated on $\tau \in T_p$ and destination track $\in T_r$
    then
38:             return 2*Reward.Park
39:         end if
40:     else if train is not to be combined and service tasks completed and origin track $\in T_p$ then
41:         return -Reward.Park
42:     else if train is to be combined and service tasks completed and origin track $\in T_p$ then
43:         if train to be combined with is being serviced or has not arrived yet at the shunting yard then
44:             return -Reward.Park
45:         end if
46:     end if
47: end if
```

Algorithm 4 Heuristics for Assignment of Rewards with IRTS-Based Routing Strategy in Heerlen Pseudocode in TORS-MATDRL (Part 2)

```
1: if action is *Service* and assigned service tasks are not completed then
2:     return Reward.Start_clean
3: else if event is *EndService* and action is *Move* then
4:     if unit does not correspond to the event then
5:         if origin track $\notin T_p$ and destination track $\in T_p$ then
6:             List trains parked on destination track $\tau$
7:             if there is at least one unit parked on $\tau \in T_p$ then
8:                 if train arrives from A-side then
9:                     if train parked in the back of the queue is to be combined with arriving train then
10:                        return 2*Reward.Park
11:                    else if train parked in the back of queue is to be combined but not with arriving train then
12:                        return -2*Reward.Park
13:                    else if arriving train departs later than train parked in the back of the queue then
14:                        return Reward.Park
15:                    else if arriving train departs earlier than train parked in the back of the queue then
16:                        return -2*Reward.Park
17:                    end if
18:                else if train arrives from B-side then
19:                    if train parked in the front of the queue is to be combined with arriving train then
20:                        return 2*Reward.Park
21:                    else if train parked in the front of queue is to be combined but not with arriving train then
22:                        return -2*Reward.Park
23:                    else if arriving train departs earlier than train parked in the front of the queue then
24:                        return Reward.Park
25:                    else if arriving train departs later than train parked in the front of the queue then
26:                        return -2*Reward.Park
27:                    end if
28:                end if
29:            end if
30:        else if origin track $\in T_p$ then
31:            List trains parked on origin track $\tau \in T_p$
32:            if train is parked in the front of the queue and is to be combined and is next to depart and destination track $\in T_g$ and 3 triggers before its scheduled departure then
33:                return Reward.Park
34:            else if the queue on $\tau \in T_p$ consists of more than one unit and parking order is violated on $\tau \in T_p$ and destination track $\in T_r$ then
35:                return 2*Reward.Park
36:            end if
37:        else if train is not to be combined and service tasks completed and origin track $\tau \in T_p$ then
38:            return -Reward.Park
39:        else if train is to be combined and service tasks completed and origin track $\in T_p$ then
40:            if train to be combined with is being serviced or has not arrived yet at the shunting yard then
41:                return -Reward.Park
42:            end if
43:        end if
44:    end if
45: else if action is *Wait* and unit is on $\tau \in T_g$ then
46:    return -Reward.Park
47: end if
```

Algorithm 4 Heuristics for Assignment of Rewards with IRTS-Based Routing Strategy in Heerlen Pseudocode in TORS-MATDRL (Part 3)

```
1: if event is EndService and unit corresponds to the event and action is SetBack then
2:     [Reward.End_clean, 0]
3: else if event is EndService and action is Move then
4:     if unit corresponds to the event then
5:         List trains parked on origin track $\tau \in T_p$
6:         if the queue on $\tau \in T_p$ consists of more than one unit and parking order is violated on $\tau \in T_p$ and destination track $\in T_r$ then
7:             return Reward.End_clean + 2*Reward.Park
8:         else if the queue on $\tau \in T_p$ consists of more than one unit and train is to be combined and origin track $\in T_p$ and destination track $\in T_r$ then
9:             if train to combine with is not in the queue on $\tau \in T_p$ or has not arrived at shunting yard yet then
10:                return Reward.End_clean + Reward.Park
11:            end if
12:        else if destination track $\in T_g$ or $\in T_r$ then
13:            return -Reward.Park
14:        else if destination track $\in T_p$ then
15:            return Reward.End_clean
16:        end if
17:    end if
18: else if event is PrepDeparture and action is Move and train is next to depart then
19:    if destination track $\in T_g$ then
20:        return Reward.Move_depart
21:    end if
22: else if action is Depart and service tasks are completed then
23:    return Reward.Leave_clean
24: else if action is Depart then
25:    return Reward.Leave_dirty
26: else if current event or next event is Departure and action is Move and service tasks are completed and origin track $\notin T_g, T_r$ then
27:    return -Reward.Park
28: end if
29: return 0
```

Algorithm 5 Heuristics for Assignment of Rewards with TBS-Based Routing Strategy in Kleine Binckhorst Pseudocode in TORS-MATDRL (Part 1)

```
Input: train unit $u \in U$, action chosen, event trigger, list of agents.
Output: reward.
1: if the action completed the instance successfully then
2:     return Reward.All_left
3: else if action is Connect then
4:     if unit is on parking track $\in T_p$ then
5:         if connection will result in the correct composition for the next departure then
6:             return Reward.Comb_correct
7:         else if unit is not in a correct composition then
8:             return Reward.Comb_correct
9:         end if
10:    else
11:        return -Reward.Comb_correct
12:    end if
13: else if action is Split then
14:    if unit is not in the correct composition and is parked on relocation track $\in T_r$ then
15:        return Reward.Split_correct unit is in the correct composition for the next departure
16:        return -Reward.Split_correct
17:    else if at least one of the involved units will be in incorrect composition after the operation then
18:        return -Reward.Split_correct
19:    else
20:        return -Reward.Split_correct
21:    end if
22: else if event is Middle or EndService with unit not corresponding to the event and action is Move then
23:    if origin track $\in T_g$ and destination track $\in T_e$ then
24:        if facility tracks $\in T_s$ are at capacity then
25:            return Reward.Park
26:        else if unit has service tasks assigned at facility tracks $\in T_s$ and facility tracks $\in T_s$ have sufficient capacity then
27:            return -Reward.Park
28:        else if route to relocation tracks $\in T_r$ is cleared and unit does not have service tasks assigned at facility tracks $\in T_s$ then
29:            return -Reward.Park
30:        else if route to relocation tracks $\in T_r$ is not cleared and unit does not have service tasks assigned at facility tracks $\in T_s$ then
31:            return Reward.Park
32:        end if
33:    else if origin track $\in T_e$ and destination track $\in T_s$ and unit has no service tasks assigned at facility tracks $\in T_s$ then
34:        return Reward.Park
35:    else if origin track $\in T_s$ or $T_e$ and destination track $\in T_r$ and unit has no service at facility tracks $\in T_s$ then
36:        if there is at least one unit parked on relocation track $\in T_r$ then
37:            return -Reward.Park*2
38:        else if there are no units parked on relocation track $\in T_r$ then
39:            return Reward.End_clean + Reward.Park
40:        end if
41:    else if origin track $\in T_r$ and destination track $\in T_p$ and there is at least one unit parked on destination track $\tau$ then
42:        List trains parked on destination track
43:        if train parked in the back of the queue is of the same type as arriving unit then
44:            return Reward.Park
45:        else
46:            return -2*Reward.Park
47:        end if
48:    else if origin track $\in T_r$ and destination track $\in T_p$ and destination track is empty and there is no unit to park next to correctly
       according to IRTS $\forall \tau \in T_p$ then
49:    else if origin track $\in T_g$ and destination track $\in T_r$ and unit has service at facility tracks $\in T_s$ then
50:        return -Reward.Park
51:    else if origin track $\in T_g$ and destination track $\in T_p$ then
52:        return -Reward.Park*2
53:    else if origin track $\in T_e$ and destination track $\in T_g$ then
54:        return -Reward.Park*2
55:    else if origin track $\in T_r$ and destination track $\in T_e$ or $T_s$ or $T_g$ then
56:        return -Reward.Park*2
57:    else if origin track $\in T_p$ then
58:        if train is not to be combined and service tasks completed then
59:            return -Reward.Park*2
60:        else if train is to be combined and service tasks completed then
61:            if train to be combined with is being serviced or has not arrived yet at the shunting yard then
62:                return -Reward.Park
63:            end if
64:        end if
65:    end if
66: end if
```

Algorithm 5 Heuristics for Assignment of Rewards with TBS-Based Routing Strategy in Kleine Binckhorst Pseudocode in TORS-MATDRL (Part 2)

```
1: if action is Service and assigned service tasks are not completed then
2:     if unit has service tasks assigned at facility tracks ∈ $T_s$ and location for selected servicing is in ∈ $T_s$ then
3:         return Reward.Start_clean
4:     else if unit has service tasks assigned at facility tracks ∈ $T_s$ and location for selected servicing is not in ∈ $T_s$ then
5:         return 0
6:     else if unit has no service tasks assigned at facility tracks ∈ $T_s$ and location for selected servicing is not in ∈ $T_s$ then
7:         return Reward.Start_clean
8:     else if unit has no service at facility tracks ∈ $T_s$ and location for selected servicing is in ∈ $T_p$ and unit is on ∈ $T_s$ or $T_r$ then
9:         return -Reward.Start_clean
10:    else if unit has service at parking tracks ∈ $T_p$ or $T_e$ and location for selected servicing is in ∈ $T_p$ and unit is on ∈ $T_g$ then
11:        return -Reward.Start_clean
12:    else if unit has service at parking tracks ∈ $T_p$ or $T_e$ and location for selected servicing is in ∈ $T_p$ and unit is on ∈ $T_r$ then
13:        List trains parked on destination track
14:        if train parked in the front of the queue is to be combined with arriving train then
15:            return Reward.Start_clean
16:        else if train parked in the front of queue is to be combined but not with arriving train then
17:            return -2*Reward.Park
18:        else if arriving train departs earlier than train parked in the front of the queue then
19:            return Reward.Start_clean
20:        else if arriving train departs later than train parked in the front of the queue then
21:            return -2*Reward.Park
22:        end if
23:    end if
24: else if event is EndService and unit corresponds to the event and action is SetBack then
25:    Reward.End_clean
26: else if event is EndService and action is Move and unit corresponds to the event then
27:    List trains parked on origin track $\tau \in T_p$
28:    if unit has pending service tasks then
29:        if origin track ∈ $T_s$ and destination track ∈ $T_r$ then
30:            return 2*Reward.Park
31:        else if origin track ∈ $T_p$ and destination track ∈ $T_r$ then
32:            return -2*Reward.Park
33:        end if
34:    else if unit has no pending service tasks then
35:        if origin track ∉ $T_s$ then
36:            return -Reward.Park
37:        else if destination track ∈ $T_g$ or $T_e$ then
38:            return -Reward.Park
39:        else if origin track ∈ $T_e$ or $T_s$ and destination track ∈ $T_r$ then
40:            if there is at least one unit parked on relocation track ∈ $T_r$ then
41:                return -2*Reward.Park
42:            else if there are no units parked on relocation track ∈ $T_r$ then
43:                return Reward.End_clean + Reward.Park
44:            end if
45:        end if
46:    end if
47: else if event is PrepDeparture and action is Move and train is next to depart then
48:    if origin track ∈ $T_p$ and destination track ∈ $T_g$ then
49:        return Reward.Move_depart
50:    else if origin track ∈ $T_r$ and destination track ∈ $T_g$ via $T_p$ then
51:        return Reward.Move_depart
52:    else
53:        return -2*Reward.Park
54:    end if
55: else if action is Depart and service tasks are completed then
56:    return Reward.Leave_clean
57: else if action is Depart then
58:    return Reward.Leave_dirty
59: else if current event or next event is Departure and action is Move and service tasks are completed and origin track ∉ $T_g$, $T_r$ then
60:    return -Reward.Park
61: end if
62: return 0
```

Algorithm 6 Heuristics for Assignment of Rewards with IRTS-Based Routing Strategy in Kleine Binckhorst Pseudocode in TORS-MATDRL (Part 1)

```
Input: train unit $u \in U$, action chosen, event trigger, list of agents.
Output: reward.
1: if the action completed the instance successfully then
2:     return Reward.All_left
3: else if action is Connect and unit is not in the correct composition and connection will result in the correct composition then
4:     if unit is on parking track $\in T_p$ then
5:         return Reward.Comb_correct
6:     else
7:         return -Reward.Comb_correct
8:     end if
9: else if action is Split and unit is not in the correct composition then
10:     if unit is on relocation track $\in T_r$ then
11:         return Reward.Split_correct
12:     else
13:         return -Reward.Split_correct
14:     end if
15: else if event is Middle or EndService with unit not corresponding to the event and action is Move then
16:     if origin track $\in T_g$ and destination track $\in T_e$ then
17:         if facility tracks $\in T_s$ are at capacity then
18:             return Reward.Park
19:         else if unit has service tasks assigned at facility tracks $\in T_s$ and facility tracks $\in T_s$ have sufficient capacity then
20:             return -Reward.Park
21:         else if route to relocation tracks $\in T_r$ is cleared and unit does not have service tasks assigned at facility tracks $\in T_s$ then
22:             return -Reward.Park
23:         else if route to relocation tracks $\in T_r$ is not cleared and unit does not have service tasks assigned at facility tracks $\in T_s$ then
24:             return Reward.Park
25:         end if
26:     else if origin track $\in T_e$ and destination track $\in T_s$ and unit has no service tasks assigned at facility tracks $\in T_s$ then
27:         return Reward.Park
28:     else if origin track $\in T_s$ or $T_e$ and destination track $\in T_r$ and unit has no service at facility tracks $\in T_s$ then
29:         if there is at least one unit parked on relocation track $\in T_r$ then
30:             return -Reward.Park*2
31:         else if there are no units parked on relocation track $\in T_r$ then
32:             return Reward.End_clean + Reward.Park
33:         end if
34:     else if origin track $\in T_r$ and destination track $\in T_p$ and there is at least one unit parked on destination track $\tau$ then
35:         List trains parked on destination track
36:         if train parked in the back of the queue is to be combined with arriving train then
37:             return 2*Reward.Park
38:         else if train parked in the back of queue is to be combined but not with arriving train then
39:             return -2*Reward.Park
40:         else if arriving train departs earlier than train parked in the back of the queue then
41:             return Reward.Park
42:         else if arriving train departs later than train parked in the back of the queue then
43:             return -2*Reward.Park
44:         end if
45:     else if origin track $\in T_r$ and destination track $\in T_p$ and destination track is empty and there is no unit to park next to correctly according to IRTS $\forall \tau \in T_p$ then
46:     else if origin track $\in T_g$ and destination track $\in T_r$ and unit has service at facility tracks $\in T_s$ then
47:         return -Reward.Park
48:     else if origin track $\in T_g$ and destination track $\in T_p$ then
49:         return -Reward.Park*2
50:     else if origin track $\in T_e$ and destination track $\in T_g$ then
51:         return -Reward.Park*2
52:     else if origin track $\in T_r$ and destination track $\in T_e$ or $T_s$ or $T_g$ then
53:         return -Reward.Park*2
54:     else if origin track $\in T_p$ then
55:         if train is not to be combined and service tasks completed then
56:             return -Reward.Park*2
57:         else if train is to be combined and service tasks completed then
58:             if train to be combined with is being serviced or has not arrived yet at the shunting yard then
59:                 return -Reward.Park
60:             end if
61:         end if
62:     end if
63: end if
```

Algorithm 6 Heuristics for Assignment of Rewards with IRTS-Based Routing Strategy in Kleine Binckhorst Pseudocode in TORS-MATDRL (Part 2)

al2:irts:drl:kleineb2

```
 1: if action is Service and assigned service tasks are not completed then
 2:     if unit has service tasks assigned at facility tracks ∈ $T_s$ and location for selected servicing is in ∈ $T_s$ then
 3:         return Reward.Start_clean
 4:     else if unit has service tasks assigned at facility tracks ∈ $T_s$ and location for selected servicing is not in ∈ $T_s$ then
 5:         return 0
 6:     else if unit has no service tasks assigned at facility tracks ∈ $T_s$ and location for selected servicing is not in ∈ $T_s$ then
 7:         return Reward.Start_clean
 8:     else if unit has no service at facility tracks ∈ $T_s$ and location for selected servicing is in ∈ $T_p$ and unit is on ∈ $T_s$ or $T_r$ then
 9:         return -Reward.Start_clean
10:     else if unit has service at parking tracks ∈ $T_p$ or $T_e$ and location for selected servicing is in ∈ $T_p$ and unit is on ∈ $T_g$ then
11:         return -Reward.Start_clean
12:     else if unit has service at parking tracks ∈ $T_p$ or $T_e$ and location for selected servicing is in ∈ $T_p$ and unit is on ∈ $T_r$ then
13:         List trains parked on destination track
14:         if train parked in the front of the queue is to be combined with arriving train then
15:             return Reward.Start_clean
16:         else if train parked in the front of queue is to be combined but not with arriving train then
17:             return -2*Reward.Park
18:         else if arriving train departs earlier than train parked in the front of the queue then
19:             return Reward.Start_clean
20:         else if arriving train departs later than train parked in the front of the queue then
21:             return -2*Reward.Park
22:         end if
23:     end if
24: else if event is EndService and unit corresponds to the event and action is SetBack then
25:     Reward.End_clean
26: else if event is EndService and action is Move and unit corresponds to the event then
27:     List trains parked on origin track $\tau \in T_p$
28:     if unit has pending service tasks then
29:         if origin track ∈ $T_s$ and destination track ∈ $T_r$ then
30:             return 2*Reward.Park
31:         else if origin track ∈ $T_p$ and destination track ∈ $T_r$ then
32:             return -2*Reward.Park
33:         end if
34:     else if unit has no pending service tasks then
35:         if origin track ∉ $T_s$ then
36:             return -Reward.Park
37:         else if destination track ∈ $T_g$ or $T_e$ then
38:             return -Reward.Park
39:         else if origin track ∈ $T_e$ or $T_s$ and destination track ∈ $T_r$ then
40:             if there is at least one unit parked on relocation track ∈ $T_r$ then
41:                 return -2*Reward.Park
42:             else if there are no units parked on relocation track ∈ $T_r$ then
43:                 return Reward.End_clean + Reward.Park
44:             end if
45:         end if
46:     end if
47: else if event is PrepDeparture and action is Move and train is next to depart then
48:     if origin track ∈ $T_p$ and destination track ∈ $T_g$ then
49:         return Reward.Move_depart
50:     else if origin track ∈ $T_r$ and destination track ∈ $T_g$ via $T_p$ then
51:         return Reward.Move_depart
52:     else
53:         return -2*Reward.Park
54:     end if
55: else if action is Depart and service tasks are completed then
56:     return Reward.Leave_clean
57: else if action is Depart then
58:     return Reward.Leave_dirty
59: else if current event or next event is Departure and action is Move and service tasks are completed and origin track ∉ $T_g$, $T_r$ then
60:     return -Reward.Park
61: end if
62: return 0
```

C
List of Business Rules

The *business rules* describe the constraints on train shunting, which can be configured to satisfy the desired functionality of TORS. These can be *hard* constraints, which cannot be violated under any circumstance, or *soft*, which implies that the actions will not be rejected if it does not meet the criteria but a penalty may be given instead. Also, business rules are given a priority, which sets the order in which they will be evaluated. Furthermore, the business rules are classified in the following categories: matching, arrival and departure, combining and splitting, service tasks, parking, moving, track occupation, facilities and personnel (the last is ruled out of this research). In order to improve the performance of the simulation, the ordering of business rules can be optimized by changing the order in which they are evaluated.

Firstly, we present the business rules for matching:

- If a train unit in an outgoing shunting unit contains a train unit id, then only the train unit with that id can be matched to it.
- If a train unit in an outgoing shunting unit only specifies a train unit type, then every train unit with that type can be matched to it.
- Only if no train unit can be found such that it can be matched to a train unit in an outgoing shunting unit, this train unit may be ignored.

Secondly, the business rules related to arrival and departure are as follows:

- The order in which outstanding shunting units are positioned on a track is defined by its index as specified in the scenario, where the shunting unit with a smaller index is positioned at the A-side relative to those with a greater index.
- An arrival will never occur before its arrival time as specified in the scenario.
- A shunting unit that is already present before the planning period will be placed as specified in the scenario. A shunting that is still present at the end of a planning period will also be parked at the specified track.
- The train units of a leaving shunting unit are present in the order in which they are specified in the scenario.
- Arrivals and departures occur at the exact times as specified in the scenario.

Thirdly, the business rules associated with combining and splitting are as follows:

- Combining and splitting shunting units does not change the order in which they are present.
- Combining and splitting shunting units occurs at a track where it is allowed to setback.
- Combining and splitting takes at least as much time as is specified in the scenario.

- Combining and splitting takes precisely as much time as is specified in the scenario.
- Combining and splitting shunting units occurs at a track where it is allowed to park.

Fourthly, the business rules related to the service tasks are the following:

- All required facilities are assigned to a service task, as is specified in the scenario. Only one facility can be assigned when a service task is executed.
- All required personnel are assigned to a service task, as is specified in the scenario.
- A service task is executed at a track where all the required facilities are available for use.
- Each train unit has finished all mandatory service tasks before it departs from the shunting yard, or the planning period is over.
- Service tasks take at least as long as specified in the scenario.
- Each train unit has finished all optional service tasks before it departs from the shunting yard, or the planning period is over.
- If a shunting unit only contains train units which are electrically powered, all service tasks on its train units are executed on electrified tracks. Checks if such shunting unit can park at a service location.

Fifthly, the business rules associated with parking of trains are as follows:

- A train can only park on a track where it is legal to setback.
- A train can only park on a track where it is legal to park.
- If a shunting unit only contains train units which are electrically powered then it can only park on electrified tracks.

Besides, the business rules related to moving trains are the following:

- The duration of moving is given by the following formula: $t = A + B \times$ *number of tracks* $+ C \times$ *number of switches*, where the values of A, B and C are provided in the scenario.
- If two drivers are assigned to a setback task, then the duration of this task is at least the standard time. The standard time it takes to setback is equal to the time it takes to setback as is specified for the first train unit in the direction of the setback as is given in the scenario. If one driver is assigned to a setback task, the duration of this task is at least equal to *standard duration + (walking time per train unit × number of train units)*.
- A shunting unit is allowed to leave in both directions when in neutral state, for example after a setback or service task was executed, but it may not make another setback again when it is already in neutral state.
- A shunting unit can only setback on a track where it is legal to setback.
- If a shunting unit only contains train units which are electrically powered, it can only move over electrified tracks.

Furthermore, the business rules relating to track occupation are the following:

- If two movement actions use the same infrastructure (for example a track or a switch), then these movements may not happen simultaneously. The only exception to this rule is when two trains depart from the same track in different directions, in which case the movements may overlap in time.
- Except for the start and the final track part, each track part over which a movement takes place must not contain other train units.
- The sum of the lengths of train units on a track part may never exceed the length of that track part.

- A movement on a shunting unit may only start if no train units are present between the shunting unit and the departure direction of the track this train is currently on. For example, if a track contains the following train units A: 4201, 4202 :B; then 4201 may not depart towards the B-side until 4202 has left.
- No tasks may utilize a track part which is disturbed or used by non-service traffic.

Lastly, the business rules related to facilities are the following:

- Tasks assigned to a facility begin and end during the operating period of that facility.
- A facility may never be used for more tasks simultaneously than its maximum capacity permits, as described in the scenario.
- No service tasks can be executed at disturbed facilities.