MASTER

Automatic algorithm configuration with search heuristics for the Train Unit Shunting Problem

van den Nieuwelaar, L.J.

Award date:
2021

Department of Industrial Engineering and Innovation Sciences
Information System Group

Automatic algorithm configuration with search heuristics for the Train Unit Shunting Problem

Ludo van den Nieuwelaar (1383264)

Supervisors:
Dr. Y. (Yingqian) Zhang - TU/e
Dr. Ir. L. (Laurens) Bliek- TU/e
Dr W. J (Wan-Jui) Lee - NS Reizigers

In partial fulfillment of the requirements for the degree Master of science in Operations Management and Logistics

Eindhoven, September 2021

Abstract

The challenges encountered when creating a planning for the shunting yards of the Dutch Railways (NS) are referred to as the Train Unit Shunting Problem (TUSP). Reinforcement learning methods in the TUSP can create shunting schedules online. By doing so, reinforcement learning overcomes the rescheduling of a solution when disturbances occur. These reinforcement learning methods inspired an approach to create an online planner that creates a shunting plan in a similar way, while executing. However, the mechanism that determines which action to take, the reinforcement learning agent, is replaced with a set of deterministic rules, referred to as the deterministic agent. This deterministic agent is not creating a number of feasible shunting plans that are satisfactory. To increase the applicability of the online creation of shunting schedules, this research considers a framework that can be used to increase the number of feasible shunting plans created by the deterministic agent. The framework consists of two sequential steps that first tries to find the optimal algorithm configuration through an iterative Bayesian Optimization approach. In this step, we analyze the impact of problem-related contextual information to increase the applicability of hyperparameter optimization techniques. The second step will consist of a Monte Carlo Tree Search (MCTS) to handle the stochasticity that occurs from analyzing different scenarios with changing arrival and departure sequences. This study shows that the Bayesian Optimization techniques that do not use contextual information show a poor generalization from the training data to the unencountered test data. When considering contextual information, the Bayesian Optimization could generalize the training data to the not encountered test data. This generalization problem occurs when the data is not homogeneous enough. Then the surrogate model is not able to detect complexity differences within the contextual information. The results of the Bayesian Optimization lead to an improved hyperparameter configuration for one group of contextual information. We also show that the number of feasible scenarios can increase when applying MCTS techniques, especially on a carousel shunting yard. However, this improvement increases computation times.

Executive summary

Research problem

This research project is executed in collaboration with the Dutch Railways, or NS. The NS operates more than 5,000 train trips through which 1.3 million passengers are transported daily. During off-peak and night hours, the carriages have to be prepared for another shift at the shunting yards. Creating a movement and service planning for a shunting yard is a challenge that is in the literature referred to as the *Train Unit Shunting Problem* (TUSP) (Kroon et al., 2008).

Reinforcement learning approaches within the TUSP create a shunting plan online by keeping track of the actions while executing the shunting plan. By doing so, reinforcement learning overcomes the rescheduling of a solution when disturbances occur. These reinforcement learning methods inspired an approach to create an online planner that creates a shunting plan in a similar way, while executing. But then, the mechanism that determines which action to take is replaced with a set of deterministic rules. The actions that can be selected through these deterministic rules are the *Departure, Split, Connect, Setback, Wait, Service* and *Move* actions. All viable actions per train are assigned a priority value through deterministic rules and the action with the highest priority value will be executed on the simulation model. For simplicity is this method referred to as the *deterministic agent.* This deterministic agent is in a state of development that it can create shunting plans. However, the deterministic agent is not creating a number of feasible shunting plans that is satisfactory.

To increase the applicability of the online creation of shunting schedules and to increase the number of feasible shunting plans, a framework is designed that consists of two improvement steps. The first step focuses on configuring the deterministic rules of the deterministic agent such that the most feasible shunting plans are generated. The priority values that are assigned to each of the actions remain fixed when executing the algorithm. Therefore these priority values of all the actions be considered as hyperparameters. Changing the hyperparameters will change the dynamics of the algorithm since an action becomes less or more important than other actions. To increase the applicability of hyperparameter optimization techniques, we address the following problem: design a hyperparameter optimization approach to configure a deterministic agent that focuses on maximizing the number of feasible solutions.

The second step of the framework focuses on the details of the deterministic rules of the deterministic agent. The deterministic rules cover a large part of the actions that can occur in the simulation model. However, some movement actions cannot be approached in a deterministic manner since these actions depend on the composition of future arrival or departure sequences. Then, the deterministic rules are not able to distinguish which track is better and makes an arbitrary choice. We show in the second step of our framework that a heuristic search algorithm is a suitable technique to overcome the stochasticity that arises from analyzing different problem instances with the deterministic rules of the agent.

To increase the understanding of our framework and thereby creating an increase in the applicability of the online creation of shunting schedules. Our optimization framework will be applied on a carousel shunting yard and a shuffleboard shunting yard.

Research approach

This thesis aims to improve the online planning capabilities of the deterministic agent by applying the recent advancements in hyperparameter optimization techniques to find the optimal algorithm configuration and a heuristic search algorithm to handle stochasticity. This research is guided by the following central question;

> *How can a framework of a hyperparameter optimization approach and a search algorithm optimize an online planning heuristic to increase the number of feasible shunting plans of parking-, routing- and service scheduling- subproblems of the Train Unit Shunting Problem?*

In this project, the formulated problem will be formulated as a Constraint Satisfaction Problem (CSP). The main difference between a CSP and a typical optimization problem is that CSP tries to generate a feasible solution with the given constraints instead of an optimal solution regarding some objective function. The first step of the proposed solution framework consists of a Sequential Model-Based Optimization technique that tries to construct a mapping between a set of hyperparameters and the resulting number of feasible solutions. This mapping will eventually be used to select the best hyperparameter configuration. The second part of the framework introduces a search strategy to provide the deterministic agent with additional information. The search strategy that will be used is the Monte Carlo Tree Search (MCTS) because this search strategy has proven to work in sequential domains (Silver et al., 2016). The MCTS follows a tree structure where all the nodes represent the states and the arcs represent the movement actions. Every MCTS iteration adds a new trajectory and the success rate of that trajectory to the tree, which eventually leads to a Monte Carlo estimate that can be used to select the optimal movement action.

Solution methods

To solve the earlier described challenges, a framework is created that consists of two solution methods. The first solution method consists of a design of the Sequential Model-Based Optimization (SMBO). The SMBO approach will consist of an iterative Bayesian Optimization technique that will be placed as a wrapper over the simulation environment with the deterministic agent. This Bayesian Optimization method will create a mapping from the selected hyperparameters to the returned objective function. This is also known as a surrogate model. Subsequently, this surrogate model will be used by an acquisition function to determine the best hyperparameter configuration to sample next. The response surface model is modeled through Gaussian Processes and the most promising hyperparameter configurations are chosen through a technique that is called Gaussian Process Upper Confidence Bound. To create a shunting schedule from a scenario, the simulation environment only uses a single logical core of a PC. Therefore, the Bayesian Optimization wrapper is formulated in such a way that multiple shunting plans are created in parallel. Roughly taken, this decreased the computation times with a factor m where m is equal to the number of logical cores of the used machine, which is eight in our research.

The second solution method is the implementation of an MCTS to handle the stochasticity. The stochasticity only occurs with movement actions which means that the simulation model contains redundant information for the MCTS. Therefore, an MCTS state space is formulated that only considers the tracks where stochasticity occurs. The MCTS states are composed by dividing the considered tracks, based on the smallest train, into dedicated parking spots which all together form an MCTS state. Every time a train is parked on one of these tracks, the state changes from s to s'. The actions that are responsible for this change in states are defined as the individual tracks since the trains are parked consecutively. Then, MCTS iterations will create trajectories by sequentially assigning trains to tracks that create a parking composition. After all the trains in the arrival sequence are parked on the considered tracks is checked if the trains can leave the considered tracks unobstructed. Then, the solution is feasible and is backed up in a matrix that consists of the parking spots and all the trains in the arrival sequence. Eventually, this matrix will be filled with Monte Carlo estimates that will be used as additional information for the deterministic agent. The matrix with the Monte Carlo estimates is recalculated after every time a train is parked on the tracks. In this way, the best parking position can be determined for every train at every moment.

This framework is tested through experiments for two types of shunting yards, the carousel and shuffleboard shunting yards. For every type of shunting yard, three associated types of scenarios are formulated that all contain different complexities.

Results

The proposed framework consists of two steps. The first step consists of the hyperparameter optimization approach through Bayesian Optimization. The first two experiments for the Bayesian Optimization empirically determines the number of iterations required for convergence and the batch size that prevents over-tuning the hyperparameters. These experiments show that the speed of convergence can be increased by using a warm starting approach where the hyperparameters of the original design of the deterministic agent are used as initial iteration. To prevent the model from over tuning on similar training data, a new batch of training data is sampled after every Bayesian Optimization iteration. After that, experiments are executed that compare the hyperparameters optimized with Bayesian Optimization with the hyperparameter of the original design of the deterministic agent. These experiments consist of two types of approaches. Namely, one set of experiments that does not consider any contextual information and one set with experiments that does consider contextual information. The contextual information in these experiments is the deviation of scenarios based on, among others, the number of train units. The experiments that do not consider any contextual information show a poor generalization from best obtained hyperparameters on the training data to the unencountered test data. The poor generalization occurs more often when the surrogate model is not able to detect variations in the underlying complexity of the contextual information (Char et al., 2019). This means that when considering contextual information, the Bayesian Optimization should be better able to generalize the solutions. After these experiments, it became apparent that the surrogate model was better able to generalize the best found hyperparameter configuration obtained on the training data to the not encountered test data. However, this difference can only be called statistically significant for experiment E. Therefore can be concluded that splitting the scenarios based on contextual information makes the complexity of the scenarios more homogeneous but does not create an improved hyperparameter configuration for all instances.

The second step of the framework consists of the experiments that use the MCTS to handle the stochasticity and thereby creating more feasible shunting plans. It can be concluded that the feasibility rate increases the most from the MCTS enhancement when the shunting yard is a carousel-like shunting yard. The shuffleboard-like shunting yards are not benefiting from the extension with the MCTS. This difference is a result of the number of considered tracks on both locations because more tracks means that it is easier for the MCTS to find a good solution and also has the MCTS a better possibility to recover from earlier made mistakes. Therefore, the deterministic agent of the carousel-like shunting yard has more room for improvement. After analyzing the lacking increase in feasible shunting plans on the shuffleboard location, it became apparent that each solution approach solves a unique set of experiments. Therefore, an additional experiment is created where the different solution approaches are placed in sequence. When placing the models in sequence, the number of feasible shunting plans increases further for both locations and approaches similar feasibility rates as the benchmark algorithm for some experiments. Therefore can be concluded that there is an increase in feasible shunting plans when using the MCTS, but this comes with a cost in the form of increased computation times. Combining the optimized hyperparameters of experiment E with the MCTS leads to an additional increase in the number of feasible shunting plans.

Conclusion and recommendation

This project focuses on solving the parking-, routing- and service scheduling- subproblems of the Train Unit Shunting Problem (TUSP) for the Dutch Railways. The main objective is to investigate to what extent a framework consisting of hyperparameter optimization and heuristic search approaches can contribute to an increase in feasible shunting plans in an online scheduling approach. By doing so, this research ensures increased applicability for the creation of online shunting schedules. Furthermore, this framework demonstrated that the number of feasible solutions could be increased by applying a hyperparameter optimization technique and improving the search capabilities of the deterministic agent. An iterative Bayesian Optimization method provided a technique to optimize the configuration of an algorithm. Thereby, increasing the number of feasible shunting plans for one experiment and increasing the applicability of these hyperparameter optimization techniques. Subsequently, the MCTS introduced a method that is able to handle the stochasticity by creating Monte Carlo estimates of feasible parking positions. These estimates are used to provide the deterministic agent with additional information. By using this framework, the number of feasible shunting plans increase substantiated and is approaching the performance of the benchmark algorithm for some problem instances. However, due to the limited time frame of the master thesis, some aspects and challenges are not included in the research. Therefore, we advise the NS to explore the following aspects before using the designed approach. 1) Analyze and, if necessary, improve the quality of the shunting schedules that are being made, 2) explore other approaches to increase the number of solved instances for the shuffleboard locations, since that performance is still lacking, 3) when using these approaches, consider a computational speed-up method such as parallel computing.

Preface

This report is written as the partial fulfillment of the requirements to obtain the degree Master of Science in Operations Management and Logistics at Eindhoven University of Technology. This project is done in collaboration with the Dutch Railways (NS), and it describes my research of the past seven months on optimizing the online creation of shunting schedules for the Train Unit Shunting Problem.

With this, I would like to take the opportunity to express my appreciation towards people who provided guidance and support throughout the master thesis project. In particular, I want to thank Yingqian Zhang. I'm very grateful that you have been my mentor during my master and I want to thank you for the freedom I received to shape the direction of the thesis towards hyperparameter optimization. Besides that, I would like to thank Laurens Bliek for the clear guidance and feedback within the field of hyperparameter optimization. In addition, I would like to thank Wan-Jui Lee, my company supervisor, for being inspirational with the knowledge of the company and the Train Unit Shunting Problem and thank you for the constructive discussions.

I want to give a special thank you to Martijn Beeks and Remco Coppens who helped me improve my wring and Robbert Reijnen for his patience in answering the questions I had. Finally, I want to show my appreciation to my close friends and girlfriend for keeping me wholesome in a time where everyone is required to work from home. I especially want to express my gratitude to my parents for their relentless support, without you I could not do this.

Ludo van den Nieuwelaar
Tilburg, September 2021

Contents

List of Figures

List of Tables

Abbreviations

BO Bayesian Optimization. 14, 15, 29, 30, 51–56, 58, 59

EI Expected Improvement. 14, 35

GP Gaussian Processes. 14, 33, 34

GP-UCB Gaussian Processes - Upper Confidence Bound. 14, 35

MCTS Monte Carlo Tree Search. 16, 18, 30, 36–40, 48, 49, 60–62, 65–67, 69

NS Dutch Railways. 2, 5–9, 11, 43, 50, 70

POI Probability of improvement. 14, 35

SLT Train type: Sprinter LightTrain. 21, 43, 45

SMAC Sequential Model-based Algorithm Configuration. 14, 15, 33, 34

SMBO Sequential Model-based Optimization. 14, 29

SNG Train type: Sprinter Nieuwe Generatie. 21, 43, 45

TORS Train Maintenance and Shunting Simulator. 20–22, 29, 32–34, 36, 37, 40–42, 46–48, 50, 51

TPE Tree-Parzen Estimator. 14, 33, 34

TUSP Train Unit Shunting Problem. 3, 4, 6, 8–10, 12, 13, 15, 16, 18, 19, 54, 68

VIRM Train type: Verlengd InterRegio Materieel. 4, 21, 43, 45

Chapter 1

Introduction

1.1 Problem context

The Dutch Railways, or NS, is the largest passenger railway operator in The Netherlands. NS operates more than 5,000 train trips through which 1.3 million passengers are transported daily. During rush- and peak- hours, almost all carriages will deploy to transfer passengers. However, during off-peak- and night- hours, fewer carriages are used. During these hours, the carriages have to be prepared for another shift which happens at the service depots or also named shunting yards. The service depots execute minor maintenance and cleaning activities to enhance the comfort and safety of the passengers. Besides that, redundant trains are parked at these shunting yards. These service depots are primarily located in densely populated areas and often close to central stations.

Creating a movement and service planning for a shunting yard is a challenge because the incoming trains have to be matched to the outgoing trains, minor maintenance and cleaning activities have to be allocated, and the trains have to be parked such that the right train can leave in the right moment. The maintenance and cleaning activities can only occur on dedicated tracks, which means that the trains have to be moved during their stay in the shunting yard. These movements, and thereby all activities, are enabled through a complex routing system where each train and track has its own length and where overtaking other trains is impossible. Figure 1.1 gives a schematic representation of the shunting yard Kleine Brinckhorst.

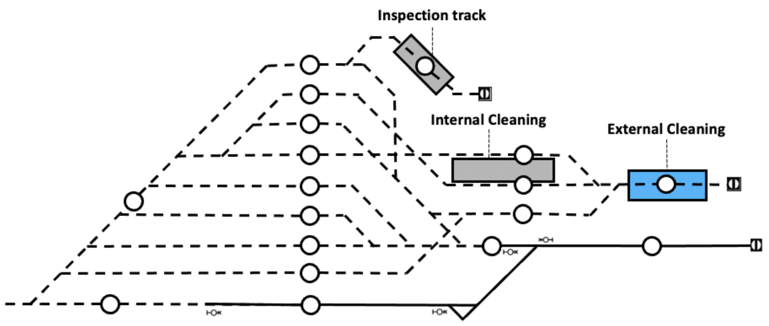


Figure 1.1: Schematic representation of shunting yard Kleine Brinckhorst (retrieved and adjusted from www.sporenplan.nl)

Currently, all activities on the shunting yards are scheduled manually by human planners. Over the upcoming years, the number of carriages will increase, meaning more trains need maintenance or cleaning. Expanding existing shunting yards or creating new shunting yards is expensive, and therefore the capacity of the existing shunting yards has to be optimized such that more trains can fit on a single shunting yard. To assist human planners in creating an optimized planning, the demand for automated planning approaches increases.

In the current literature, this scheduling problem around the shunting yard is known as the *Train Unit Shunting Problem* (TUSP) (Kroon et al., 2008). Almost all the developed approaches solve subproblems of the TUSP in an offline manner which means that these approaches can only generate entire solutions considering the subproblems (Haijema et al., 2006; Van Den Akker et al., 2008; Haahr et al., 2015; van den Broek, 2016). When such an offline approach is used, the whole solution has to be rescheduled when disturbances occur, which makes it hard for human planners to use these methods to adjust or evaluate their shunting schedules.

A more recent research direction is Reinforcement Learning. Reinforcement learning can create a shunting plan online by keeping track of the actions while executing the shunting plan and thereby overcome the rescheduling of a solution. The Reinforcement Learning direction showed promising results for some subproblems of the TUSP (Peer et al., 2018; Lee et al., 2020). These reinforcement learning methods inspired an approach to create an online planner that creates a shunting plan in a similar way, while executing. However, the mechanism that determines which action to take, the reinforcement learning agent, is replaced with a set of deterministic rules. The actions that can be selected through these deterministic rules are the *Departure, Split, Connect, Setback, Wait, Service* and *Move* actions. These deterministic rules first select the viable actions. This is done by masking the actions that are not viable. For example, a departure action is not viable and masked if the train is not on the departure track. Subsequently, all the viable actions per train are assigned a priority value through deterministic rules that could range from 'assign a value when the action is viable for this train' that occurs for departure actions to 'does the train on the ending track has to move earlier than this train' which occurs for movement actions. When all viable actions have received a priority value, the action with the highest priority value will be executed on the simulation model. These steps are repeated until all the trains in the simulation model have left the shunting yard. Then, the shunting plan is called feasible. We will refer to this method by calling it the *deterministic agent.* This deterministic agent could be used in future works as demonstration data for reinforcement learning algorithms. Demonstration data can improve the learning capabilities for reinforcement learning algorithms in large and complex environment (Hester et al., 2018). This deterministic agent is in a state of development in which it is able to generate shunting schedules. However, the deterministic agent is not creating a number of feasible shunting plans that is satisfactory.

To increase the applicability of the online creation of shunting schedules and to increase the number of feasible shunting plans a framework is designed that consist of two improvement steps. The first step focuses on configuring the deterministic rules of the deterministic agent such that an algorithm configuration is generated that creates the most feasible shunting plans. The priority values that are assigned to each of the actions remain fixed when executing the algorithm. For the location Kleine Binckhorst are the departure, split, connect, setback and wait priority values 400, 380, 280, 200 and 100 respectively[1]. This means that the

[1]The movement actions have multiple priority values based on the function of end track

departure and wait actions are the most and least important actions, respectively. Therefore can these priority values of all the actions be considered as hyperparameters. Changing the hyperparameters will change the dynamics of the algorithm since an action becomes less or more important than other actions. Ideally, all possible hyperparameter configurations are checked because this can lead to more solved instances. However, this is suboptimal considering the computational effort since a small number of hyperparameters leads to many possible configurations, which becomes difficult to evaluate considering the computation times. Therefore, model-based hyperparameter optimization techniques will be used to overcome the exhaustive computation times. Usually, hyperparameter optimization techniques are applied to optimize a solution and thereby improve the results. However, we apply the hyperparameter optimization techniques to improve the number of feasible shunting plans, which is a satisfaction problem. This is a satisfaction problem since we do not consider the quality of a problem but only if the solution is feasible. Besides that, we will investigate if contextual information related to the problems requires a generalized or a specialized hyperparameter configuration. To increase the applicability of hyperparameter optimization techniques, we address the following challenge: design a hyperparameter optimization approach to configure a deterministic agent that focuses on maximizing the number of feasible solutions.

The second step of the framework focuses on the details of the deterministic rules of the deterministic agent. Many deterministic rules cover a large part of the actions that can occur in the simulation model. However, some movement actions cannot be approached in a deterministic manner since these actions depend on the composition of future arrival or departure sequences. For example, when two parking actions are available for a single train and each action leads to an empty parking track, where both tracks have different lengths. Then, the deterministic rules are not able to distinguish which parking track is better. This could only be specified when considering the departure sequence of the trains that will be parked on these tracks in future time steps. It is possible to cover such situations with deterministic rules, but then a large number of deterministic rules has to be added to cover all possible configurations for all the problem instances. Currently, an arbitrarily choice is made when the deterministic rules are not sufficiently specified. Therefore, we show in the second step of our framework that a heuristic search algorithm is a suitable technique to overcome the stochasticity that arises from analyzing different problem instances with the deterministic rules of the agent. To increase the understanding of our framework even further and thereby creating an increase in the applicability of the online creation of shunting schedules. Our optimization framework will be applied on a carousel shunting yard and a shuffleboard shunting yard.

1.2 The Train Unit Shunting Problem

Before the TUSP is elaborated, the composition of trains and their terms will be explained. A train can consist of up to three train units, which are self-propelling vehicles that can drive in both directions. These train units have individual names, consisting of the train unit type and the number of carriages e.g. VIRM-4.

As previously illustrated, the planning problem on the shunting yard is a sequential decision-making problem referred to as the TUSP. The TUSP consist of five different subproblems based on the description of Freling et al. (2005), where the first problem is the *Matching problem*. The matching subproblem assigns incoming train units to outgoing train units. This means that a train must be split if the incoming train does not have the right composition, and subsequently, it must be combined with other train units. Besides that, the match must be feasible regarding the time on the shunting yard considering the processing times of the different maintenance and cleaning processes. In actual situations the matching is often predefined because the NS wants the right train at the right track to cope with large passenger demands in, for example, rush hours. Throughout the thesis, such a predefined matching is used.

The second subproblem is the *service scheduling*. This subproblem focuses on the maintenance and cleaning activities that have to be done on dedicated service tracks. Since there are a limited number of dedicated service tracks, the trains have to move during their stay on the yard to enable the service. This creates a complex subproblem because firstly, the service activities have to be done before the scheduled departure, and secondly, the service scheduling on dedicated service tracks implies an even more complex routing subproblem.

Regardless of a train needing service or maintenance, whenever trains are not needed for a service or maintenance job, they need to be *parked* somewhere on the shunting yard. The parking is mostly done on the dedicated parking tracks. When considering the parking problem, two major components provide lots of complexity. First, the departure sequence, a departing train must depart smoothly without any blockage or inference of other trains. Secondly, the trains need to be parked such that the lengths of the sequentially parked trains do not exceed the length of the tracks. This all while considering that trains cannot overtake each other.

Throughout the past section it can be noticed that the *routing* is another complex subproblem that influences almost all other subproblems. This is because a movement action blocks other trains from moving and a complex relocation has to be executed when one train needs to overtake another train.

Besides these problems, there is a last subproblem, the *crew scheduling* subproblem. This subproblem relates to the personnel schedule necessary to execute the movement and service activities on a shunting yard. This subproblem can be approached separately without interfering with other subproblems. Therefore, this subproblem will be disregarded throughout the master thesis.

1.3 Research questions

This thesis aims to improve the online planning capabilities of the deterministic agent by applying the recent advancements in hyperparameter optimization techniques to find the optimal algorithm configuration and a heuristic search algorithm to handle stochasticity. This research is guided through the following central question;

How can a framework of a hyperparameter optimization approach and a search algorithm optimize an online planning heuristic to improve the number of feasible shunting plans of parking-, routing- and service scheduling- subproblems of the train unit shunting problem?

This main question can be divided into multiple subquestions, which are;

1. What are the existing methods for solving the Train Unit Shunting Problem, and what are the advantages and disadvantages?

2. What is hyperparameter optimization and what are applications for satisfaction problems?
3. What is a heuristic search algorithm and what are the applications for handling stochasticity?
4. What are the characteristics of the deterministic agent for the Train Unit Shunting Problem?
5. How can an optimization framework be formulated in order to enable hyperparameter optimization and heuristic search algorithms for the deterministic agent?
6. What is the performance of such an optimization framework compared to existing techniques and how can this be explained?

1.4 Outline

The first three subquestions are answered in chapter 2 by executing a literature study. The first part of that chapter will focus on the first research question, which aims to give a broad view of the works done within the NS to solve the TUSP. This part is ordered based on the considered approaches. The second part of the literature study will focus on the hyperparameter optimization techniques and the third part of the literature study will focus on a heuristic search algorithm. Finally, this chapter is finished with a description of the academic gap the of this thesis.

The fourth research question is answered in chapter 3. Throughout this section, the problem and an initial measurement will be formulated. The fifth subquestion is answered in chapter 4 by elaborating on the integration of the improvement framework with the simulation model that contains the deterministic agent. Moreover, the described solution approach is transformed into an experimental setup. This experimental setup will be described in detail in chapter 5. The performance of the implemented framework will be elaborated in chapter 6. Besides that, this part answers the sixth subquestion. After this, the research is concluded, and the recommendations are formulated in chapter 7. A representation of the dependencies is given in Figure 1.2.

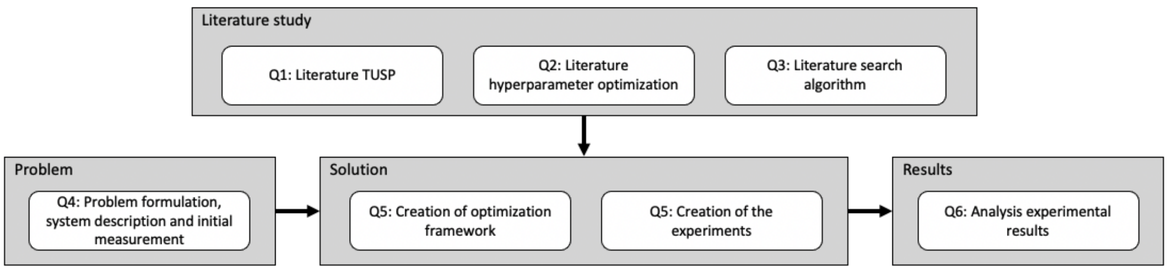

Literature study
Q1: Literature TUSP
Q2: Literature hyperparameter optimization
Q3: Literature search algorithm
Problem
Q4: Problem formulation, system description and initial measurement
Solution
Q5: Creation of optimization framework
Q5: Creation of the experiments
Results
Q6: Analysis experimental results

Figure 1.2: The composition and dependencies of the different subquestions.

1.5 Contributions

This master thesis increases the applicability of the online creation of shunting schedules and hyperparameter optimization techniques by providing insights on the following:

- We study the effects of an online planning approach that is unique to reinforcement learning combined with a deterministic control mechanism to solve the parking-, routing- and service scheduling- subproblems of the of the Train Unit Shunting Problem. This approach or the so-called 'deterministic agent', is developed by the NS and is used as a base. This base is complemented with a framework to create more feasible shunting plans. This framework consists of an approach to optimize hyperparameters to improve the dynamics of the algorithm and this framework consists of a heuristic search algorithm that is able to handle the stochasticity that can be found in different problem instances.

- We define a hyperparameter optimization approach, which is integrated into a framework to configure the prioritization of an online planning heuristics, with the objective of solving a satisfaction problem rather than an optimization problem. This approach is used for the parking-, routing- and service scheduling- subproblems of the Train Unit Shunting Problem.

- We extend our hyperparameter optimization approach to determine the influence of problem-related contextual information. By making a difference in contextual information throughout the experiments, greater insight into the generalizability of the deterministic agent is gained.

- We extend our framework to determine the influence of different types of shunting yards. Through the implementation of the framework on a shuffleboard shunting yard and a carousel shunting yard greater understanding of the online creation of shunting schedules is obtained.

Through these contributions, this work distinguishes itself and motivates a further research to the applicability of hyperparameter optimization techniques on online planning approaches.

Chapter 2

Literature Review

The goal of this literature review is to elaborate on the academic motivation to execute this research. To accomplish this is the literature review deviated into two different parts. The first part focuses on the current literature of the Train Unit Shunting Problem (TUSP) and the second part focuses on the developments of the solution methods. Both are used to formulate a clear academic gap that this research will substitute.

2.1 Train Unit Shunting Problem literature

There is much literature on the TUSP, either focusing on a specific subproblem or addressing the subproblems in an integrated manner. The following section starts with an overview of the approaches that have been used to solve the TUSP within the NS. This section can be used to create a good understanding of the approaches that have previously been applied and learn from their challenges, advantages, and disadvantages.

2.1.1 Exact solution approaches

The exact solution methods consist of approaches that can be solved through a Mixed Integer linear Programming method. These exact approaches represent a large part of the existing literature. The different subproblems that can be distinguished within the TUSP are formulated in the work of Freling et al. (2005). Besides that, these authors were the first that formulated a mathematical model to solve the matching- and parking subproblems of the TUSP with tracks that could be accessed from both sides (free tracks). The model consists of two steps, the first step matches the arriving train units to the departing train units, which solves the matching problem. Where the second step assigns trains to a specific shunt track which solves the parking problem. The other subproblems defined by the author are not taken into consideration. The authors used the two-step approach because they believed that the model would be intractable if solved at once, and with this formulation, the planners can interfere with the obtained results between both steps. In the work of Lentink et al. (2006), the model is extended with an approach to take the routing into consideration. This is done by expanding the model to a four-step model which includes a step that counts for an estimate of the routing costs between the matching and parking steps of Freling et al. (2005) and a step that counts for the final routing costs (Lentink et al., 2006). The results of the model were promising but not ready to be used because still manual modifications had to be made to the planning to be feasible in real-world scenarios.

Lentink (2006) proposed a slightly different approach to solve all the subproblems of the TUSP. The author proposed an integrated method to solve the matching and parking problem in one model. The other subproblems are solved independently with information that is passed on between the different parts. The integration of the matching and parking subproblems is further elaborated in the work of Kroon et al. (2008). Kroon et al. (2008) formulated a model based on Last In First Out tracks, Last In First Out tracks are tracks that can be reached through only one side, the other side is connected to a bumper. Such a model formulation led to an exponential number of new constraints due to the constraints that restricted crossings. Crossings occur when a train unit obstructs other train units during its departure of arrival[1]. Therefore, the model could only be solved in a reasonable time through clique constraints. The formulations in the work of Lentink (2006) and Kroon et al. (2008) served as input for the planning tool that was, at that time, used by the human planners within the NS. Another research in the matching- and parking- subproblems of the TUSP is the work of Haahr et al. (2015). Haahr et al. (2015) showed a comparison between different formulated models to solve the matching and parking problem integrally. The most interesting model is the constraint programming model. The main idea of the constrained programming model is that the composition and changes in the composition are assigned to the tracks instead of the events. This resulted in a model that does not scale well due to the high demand for memory. To speed up the model, a delayed column generation heuristic is proposed. This heuristic gives per track a set of feasible solutions, which reduces the number of variables needed. With the improvement of the heuristic, the computation time is reduced, however, this only holds for less constrained models.

In all the previous works, the service scheduling is not considered, this is done in the master thesis of van den Bogaerdt (2018). The author relaxed decision diagrams on a mathematical model for a multi-machine scheduling problem. The author used a shunting yard as a case study and modeled every track as individual machines which used a decision diagram to calculate an improved lower bound for the service scheduling problem. This model generated successful results but did not take the other aspects of the TUSP problem into consideration. Besides that, the formulated model showed its potential, but it has to be investigated further before it can be used.

The research field of the exact approaches is not improving the last years. This is because the mathematical approaches all have the same difficulties, namely minimizing the computation times. To decrease the computation time, the complexity of the models is often lowered, which means that one of the subproblems is disregarded or calculated separately, which will decrease the quality of the solution. Besides that, there are no exact solution approaches that use an online planning structure. Therefore can be concluded that these approaches are less suited for our approach since still a complete rescheduling has to take place when minor disturbances occur. The following section will elaborate more on the heuristics and metaheuristics used to solve the TUSP.

2.1.2 Heuristic solution approaches

The first heuristic that is formulated to solve the TUSP is a dynamic programming-like heuristic (Haijema et al., 2006). The authors state that the quality of a problem consists

[1]The term *crossings* is firstly introduced by Gallo and Miele (2001).

of *(I) the minimization of the movement and (de)couplings*, *(II) the grouping of train types on a shunting track is maximized* and *(III) the minimization of disturbance risks.* These problems have a conflicting nature, and therefore, the TUSP is too complex to solve optimal efficiently. The authors propose a heuristic that does not find an optimal solution but a good, robust, and practical solution. This is done by decomposing the planning horizon into smaller subproblems that relate to smaller planning periods. A planning period contains all the consecutive arrival events and all the consecutive departure events. After the planning periods are created, the planning periods are solved with a heuristic that relies on formulated business rules. This heuristic only solves one specific planning period at a time. After all the planning periods are solved, the heuristic is applied again, but now over the whole planning horizon and the earlier formulated solutions of the planning periods. After this method is tested on several test cases and a real-life scenario, the authors conclude that the heuristic should be tested more to say anything useful about the heuristic. The authors state that the model could be used to support the planners by generating a set of feasible plans where the planners have to pick the optimal or most robust solution. However, because the model heavily relies on business rules formulated by the same planners, the tool would hardly search outside the solution space that is known by the planner. Which obstructs the method to find different solutions.

Another heuristic is formulated by Van Den Akker et al. (2008). The authors describe an algorithm that is used to solve a matching and parking problem, the routing is not taken into consideration. Firstly, the greedy algorithm evaluates the planning horizon backwards and when there is a departure event, it matches the departures to a shunting track. When there is an arrival event, it is matched with a train that is currently on the shunting track. The greedy part of the algorithm comes in when the model can take 'shortcuts' to the solution by allocating a departure directly to an arrival. This greedy algorithm has the advantage that it can always find a feasible solution in a reasonable time, it can only come at the cost of extra shunting moves. However, according to the authors, the solution may not be of any practical use if not improved upon.

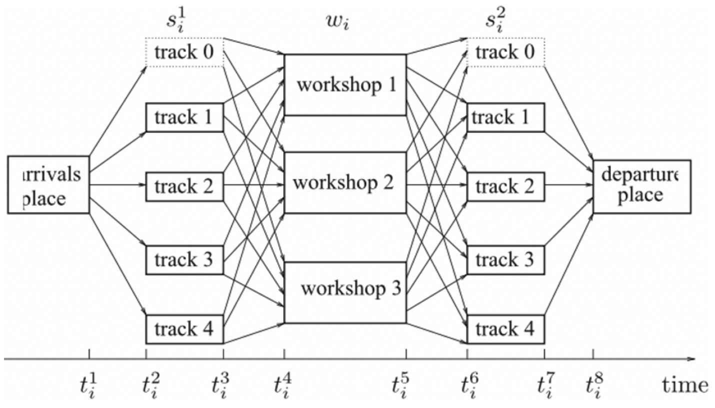


Figure 2.1: Time-space network formulation of a shunting schedule of a single train (retrieved from Jacobsen and Pisinger (2011))

Jacobsen and Pisinger (2011) formulated three metaheuristics to solve the parking- and service scheduling- subproblems in an integral fashion. The three metaheuristics, *(I) Guided Local Search*, *(II) Guided Fast Local search* and *(III) Simulated Annealing*, are used to solve a model that has multiple goals. The first goal is that trains cannot block each other, the second goal is that the trains do not depart with any delay, and finally, the makespan of the service activities is minimized. To enable these methods the shunting schedule is represented as a time-space network formulation (Figure 2.1). After the analyses, the Guided Fast Local Search is not recommended by the authors, the solving method is fast, but it has a poor performance in finding feasible solutions. The Guided Local Search and Simulated Annealing have a better performance and converge in reasonable computation times. The performance of these two algorithms is almost the same as a mathematical model which is used as a reference. This mathematical model is only focused on finding feasible solutions. Due to the slightly better computation times, the Guided Local Search is recommended over the Simulated Annealing. However, the scenario used by the authors, in both the heuristics and the mathematical model is still a simplified version of the reality.

In the work of van den Broek (2016) an integrated approach is proposed that uses a local search algorithm to solve real problem instances. The author used a simulated annealing algorithm that constructs a shunt plan which consists of the matching, service, parking and routing activities in a suboptimal order. By modeling the activities as a graph with nodes, small changes could be made through a local search algorithm without changing the structure of the graph (Figure 2.2). The results were outperforming the used Opstel Plan Generator, a tool that was used at the time to support the planners within the NS and therefore, the local search algorithm was incorporated into the software tools.

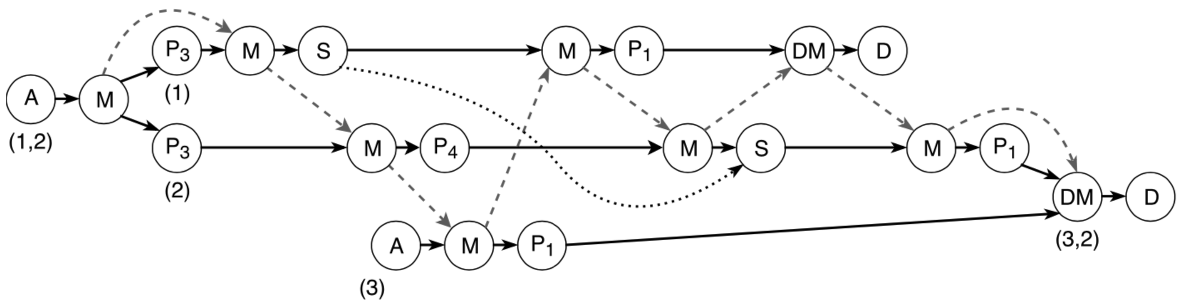


Figure 2.2: Representation of the local search graph, the arrows represent the changes the local search algorithm can make. The nodes with the letters A, M, S, D and P_i represent the arrival, movement, service tasks, departure and parking tracks, respectively (retrieved from van den Broek (2016))

The heuristic solution approaches all suffer from the same problem that as soon as disturbances occur the model has to reschedule the complete solution and thereby loses consistency. Therefore are these approaches able to generate good initial solutions but not able to generalize with disturbances. This makes it hard to use these tools for small changes in the schedule in a consistent manner because they do not conform to the logical reasoning of the human planners. The next section will elaborate on the reinforcement learning approaches which are theoretically better in generalizing the solutions.

2.1.3 Reinforcement learning solution approaches

In the work of Peer et al. (2018) the authors used a Deep Reinforcement Learning approach to solve subproblems of the Train Unit Shunting Problem. In this work, the authors used a Deep Q-Network to solve the matching- and parking- problems of the Train Unit Shunting Problem. The authors introduce experience replay to reduce the underlying correlation in combination with a visual state representation (Figure 2.3) which acts as a Markov Decision

Process. The performance of the model is measured with two measures, first the number of solvable instances, to see the performance of the model, and secondly, the entropy measure to evaluate to what extend the model uses a specific strategy. For the experiment, the authors use instances generated by an instance generator ranging from 14 to 17 trains consisting of two types of trains. The authors state that the performance of the model is good if compared with a greedy algorithm. When the performance is compared with the solution developed by van den Broek (2016) the performance in creating feasible solutions is still 10% lower. However, the feasible shunting plans are more consistent since these are created through execution of the shunting plans.

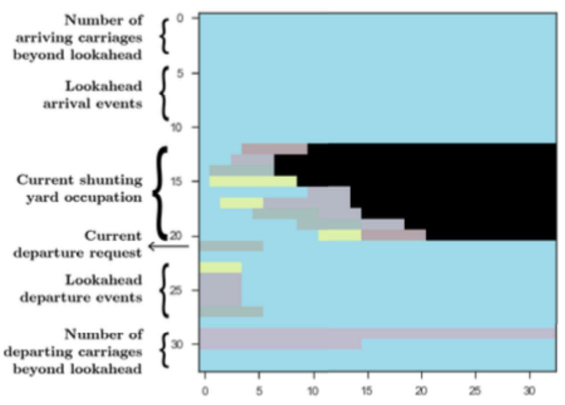

Number of arriving carriages beyond lookahead
Lookahead arrival events
Current shunting yard occupation
Current departure request
Lookahead departure events
Number of departing carriages beyond lookahead
0
5
10
15
20
25
30

Figure 2.3: Visual state representation of a state with nine tracks and departure request (retrieved from Peer et al. (2018))

Another work with the same case study is done by Lee et al. (2020). Here, the authors took besides the parking- and routing- subproblems, the service scheduling subproblems into consideration which is, according to the authors, one of the key components of the TUSP. The authors formulate different triggers which indicate agents when to take actions, such a trigger consist of a timer that counts down which indicates the time left until execution. To counteract these triggers an extra *wait* action is added to the list of possible actions. This method learns the agent the definition of time in a sequential planning problem. The size of the action list is reduced to simplify the problem, this is done by removing the combining and splitting actions. The problem is modeled as a Markov Decision Process and is solved using a Deep Q-Network Reinforcement Learning agent. Due to the constrained environment and the many available invalid actions, the agent does not end the episode when taking an invalid action. Subsequently, a large negative reward is given and the state of the agent is reset to the previous state and the policy is changed to a greedy policy. If the agents still choose an invalid action after three attempts a large negative reward is given and the episode is terminated. Besides the Deep Q-Network, another method is proposed, namely, the Value Iteration with Post-States. This method is proposed to increase the learning speed and this method shows the agent the deterministic transition function in advance. The real-world experiment is executed on a relatively small number of trains, namely 4 to 7 trains. When the results are analyzed it became apparent that the agent is capable of learning the concept of time with the triggers that can be counteracted through the wait actions. However, when the agent has to conform to many constraints there will be a trade-off between multiple constraints, therefore the authors suggest a multi-objective approach for further research.

A work that continued on Lee et al. (2020) is the bachelor thesis of Barnhoorn (2020). This work is not officially published but contains useful information for the master thesis that will be executed. In this work, a multi-agent Deep Q-Network approach is used to improve subproblems of the TUSP. The subproblems on which are being focused within this work are the matching, including recombination and the parking subproblems. The author used a multi-agent approach to reduce the number of actions that can be done by each agent, besides that a global approximation function is used for all the different agents. The states of the environment were represented through different vectors that were combined to a large vector. The results obtained in this research proved that it is possible to schedule the matching and parking subproblems with Reinforcement Learning but the performance was suboptimal on a few aspects. This is because the model diverged after a certain training period. Besides that, the complexity of input instances consisted only of four different trains, when scaling the problem to seven different trains the performance of the model dropped. However, unless the lacking performance a valuable conclusion can be made, the Multi-agent Reinforcement Learning approach allows the algorithm the opportunity to recombine different trains before leaving.

The reinforcement learning approaches have many similarities compared with the simulation environment in our thesis. In these works, the interaction with the environment and the online approach to create a shunting schedule is similar to the approach we consider in our work. However, the way of determining which actions to take differs. The similarities in approaches between our work and the reinforcement learning methods can be used to determine a suitable hyperparameter configuration method or a heuristic search algorithm by looking at successful applications in the reinforcement learning domain.

2.2 Hyperparameter optimization

The literature for the algorithm configuration and hyperparameter optimization focuses on finding the best possible set of hyperparameters regarding a certain objective function. Within the hyperparameter optimization, two broad approaches can be distinguished, namely the model-free and model-based approaches. The model-based approaches focus on creating a probabilistic surrogate model and use that model to determine the next best hyperparameter set. With a model-free approach such a model is not considered and the hyperparameter sets are tested in a random directed fashion.

2.2.1 Model-free optimization

A very simplistic model-free hyperparameter optimization algorithm is the grid search. Grid search is an exhaustive method where each hyperparameter receives multiple predefined values that need to be tested. Subsequently, all possible combinations of all hyperparameters are tested and the results are stored. After all combinations are tested the best hyperparameter configuration is chosen. This method suffers from the *curse of dimensionality* because the number of iterations increases exponentially with the number of hyperparameters. Tuning hyperparameters can be approached as an optimization problem in which more combinatorial optimization techniques can be used. Examples of combinatorial optimization techniques are Evolutionary Algorithms. Evolutionary Algorithms are algorithms that are designed with inspiration from nature, a form of an Evolutionary Algorithm is a Genetic Algorithm. Genetic Algorithms are around since 1975 but the original paper is still to be found (Srinivas and Patnaik, 1994). Genetic Algorithms consist of random searches that are directed towards an optimal solution. The base of each Genetic Algorithm are the individual solutions, named individuals, and the population is the full set of individuals. These individuals are used in an iterative procedure where the individuals are tested against a fitness function and then used to mutate and merge with other individuals such that more of the solution space will be analyzed. After that, the procedure starts over and is repeated until a computational resource is exhausted. The benefit of a Genetic Algorithm is that is almost applicable in every case.

Other Evolutionary Algorithm approaches are well suited for such optimization problems. However, the GA and similar model free approaches assume an easy-to-evaluate fitness function which will not be the case throughout this thesis since the algorithm consist of a simulation model. Therefore will the model-based approaches be formulated next.

2.2.2 Model-based optimization

Sequential Model-Based Optimization (SMBO) is a type of model-based optimization methods that tries to build a probabilistic model from an unknown objective function. This objective function is approached as a black box and subsequently, this probabilistic model is used in combination with an acquisition function to query the best possible move for the algorithm to execute next. Probably one of the most famous algorithms is the Bayesian Optimization (BO) algorithm (Močkus, 1975). The way of formulating the probabilistic model within BO can differ, throughout the literature are different probabilistic models proposed for generating this surrogate model. The most common surrogate modeling technique is based on Gaussian Processes (Rasmussen, 2003). Such a model tries to create a probabilistic model with Gaussian Processes (GP), which has the advantage that it does not lose its acquired distribution after optimizing the acquisition function. Besides that, it is possible with GP to analytically derive the acquisition function. The acquisition function determines where to query the following sample, this means that the acquisition function balances the trade-off between exploration and exploitation. Some widely used acquisition functions within GP are the Gaussian Process Upper Confidence Bound (Srinivas et al., 2009), the Probability of Improvement(Kushner, 1964) and Expected Improvement (Močkus, 1975). With the Gaussian Process Upper Confidence Bound (GP-UCB) the upper bound of the mean is calculated. This is done for all the given points and the acquisition function searches the highest valued confidence bound to place the next sample. The Probability Of Improvement (POI) considers the best-observed value function and calculates the probability, with the GP, that the best new value will exceed this initial value. The Expected Improvement (EI) will work similarly as the POI but besides the probabilistic value, the actual improvement is considered.

Besides the BO there are other well-known methods for hyperparameter optimization. One of these methods is named Sequential Model-based Algorithm Configuration (Hutter et al., 2010). The dynamics of this Sequential Model-based Algorithm Configuration (SMAC) approach are similar to the BO optimization approach but instead of formulating a probabilistic method with GP, a random forest regression method is used. Therefore, this approach is well suited to combine continuous hyperparameters with categorical parameters where BO only considers continuous values. Besides the surrogate model, SMAC can use similar acquisition functions as formulated with BO.

The last model-based hyperparameter optimization method that will be described is the Tree-Parzen Estimator (TPE) formulated by Bergstra et al. (2011). This approach creates a tree with Parzen estimators which divides the obtained samples into two different categories based on a small or large loss. Combining these trees gives a likelihood estimator to obtain the best next samples. Combining these two trees can be done with some kernel density estimators, e.g. the high loss estimates can be divided by the low loss estimates.

Contextual model-based optimization

When using model-based hyperparameter optimization, one hyperparameter set for a complete problem might be suboptimal since the problem can consist of different complexities that require different hyperparameters. Then, problem related contextual information is useful to consider. With contextual model-based optimization problem related contextual information is considered while selecting the optimal hyperparameters. When analyzing the literature of contextual hyperparameter optimization two types can be distinguished. Finding a general hyperparameter configuration for all contexts or finding a unique parameter configuration for each context individually. First, the works that try to obtain a general hyperparameter configuration for multiple contexts will be elaborated. The SMAC algorithm described in Hutter et al. (2010) differentiates between multiple problem instances but the authors try to find a configuration for all instances. In the work of Huang et al. (2006) the authors use multiple-fidelity problems which are expensive to sample. Therefore the authors create a single surrogate model to reduce the cost of the evaluation, subsequently, an augmented expected improvement method is used as an acquisition function. In this work, the authors try to create a generalized solution for multiple-fidelity optimization problems. A similar approach is taken in the work of Forrester et al. (2007). Here, the authors try to optimize a multi-fidelity problem by implementing a surrogate model of the expensive function. The objective of the authors is to obtain a solution that can be generalized over the multiple-fidelity's. The main difference with the work of Huang et al. (2006) is that the authors are using a different approach to obtain the surrogate model.

The other type of hyperparameter optimization focuses on finding an optimal set of hyperparameters for every context. First, some works are elaborated which demand a sequential approach to different contexts. In the work of Bardenet et al. (2013) the authors introduced an approach to find different configurations for a classification algorithm when applied on different data sets. This is done by introducing a ranking procedure that uses prior calculated hyperparameter configurations to find a configuration for new data sets more quickly. This research is extended in Yogatama and Mann (2014) by creating a more efficient method by assuming an online learning approach. The following methods do not require a sequential approach to different contexts. In the work of Bardenet et al. (2013) the authors introduce another approach where they apply the ranking procedure to multiple data sets simultaneously by applying one iteration on each problem in turn. Another method is introduced by Chung et al. (2020), here the authors used an offline contextual BO approach to optimize the plasma density in a nuclear fusion reactor. To do this the authors use a simulation model as a black box model which is optimized using Thompson Sampling (Thompson, 1933) as acquisition function. The offline context is generated by using different input settings which is not a part of the hyperparameter setting. The authors obtain successful results but the bottleneck of expanding this approach to further works is the expensive computation time of the simulation model. Due to the similarities in the work of Chung et al. (2020) and this thesis this work can be used as inspiration for the problem approach. Finally, while writing this thesis, another master thesis is completed that used contextual hyperparameter optimization for the TUSP van der Knaap (2021). This approach is used to optimize the hyperparameters of the Local Search approach developed by van den Broek (2016) which is described earlier. Throughout that thesis a contextual hyperparameter optimization is applied to solve a satisfaction problem. This research led to no improved in hyperparameter configuration, for any of the context since the encountered difficulties in the contextual information are too heterogeneous. They were able to improve the performance by applying a portfolios with different hyperparameters. That research will be used as inspiration for our thesis but it will not be considered to make any statements about the behavior of the models since that work is not peer reviewed.

2.3 Search algorithm

The following section will elaborate the search algorithm. The developed methods is a deviation on a reinforcement learning approach to solve the TUSP. More specifically, the sequential planning aspect of the reinforcement learning agent is used in our setup. Therefore, we have looked to search strategies that have proved to work on a reinforcement learning problems. One approach, the Monte Carlo Tree Search stands out. This approach is used in the groundbreaking research to master the game of Go with reinforcement learning (Silver et al., 2016). The Monte Carlo Tree Search (MCTS) is able to progressively expand the a tree structured search to find more solutions, these structures are often used in two player games and pathfinding algorithms. A tree search problem is characterized by states that can be represented as nodes of a tree, the arcs between the nodes represent the available actions that can be used to change between these states. In the MCTS literature three major directions can be distinguished, namely, problems with one opponent, problems with multiple opponents and problems without any opponents or also known as optimization problems (Chaslot, 2010). For our thesis only the optimization related problems are interesting and therefore these will only be discussed throughout this literature review. The MCTS consist of four steps that are executed in a iterative manner until a predefined number of iterations is reached or a computational resource is exhausted. This iterative procedure is given in Figure 2.4 and will be elaborated step by step.

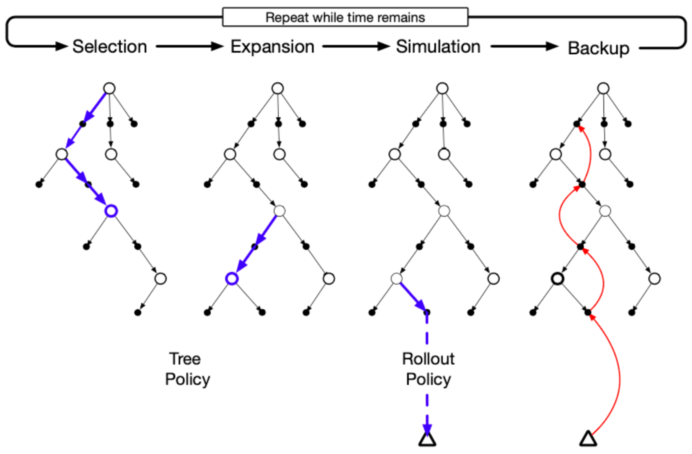


Figure 2.4: Iterative procedure of the Monte Carlo Tree Search, this iterative procedure will be executed as many times as the computational resources permits (retrieved from Sutton and Barto (2018)).

2.3.1 Selection

The first step is the selection step, this step works as follows. From the root node, which is the first node in the tree, a selection strategy is applied until a unseen node is reached. Therefore the selection step balances the exploration and exploitation trade-off. This is because it can focus on the exploration side to discover new unseen areas and it can focus on the exploitation side to exploit the best results obtained results so far. In the literature are three different strategies to handle this trade off in a structured manner, the first is the Objective Monte-Carlo method devloped by Chaslot et al. (2006). This approach consist of two parts, a move selection strategy and a backpropagation strategy. The move selection strategy uses the Central Limit Theorem to approximate the probability of a specific Monte Carlo evaluation. This approximation is subsequently used to exploit the best policy. The backpropagation strategy gives the value of a move according the values and standard deviations of the child nodes. This has the advantage that when not many simulations are made the values are close to random which is desirable since the algorithm needs to gather enough information before it chooses a direction. The more iterations are done, the more accurate the prediction becomes. The second technique is introduced by Coulom (2006), this technique also consists of a improved selection strategy and a backpropagation strategy. Here the backpropagation strategy is similar as described in Chaslot et al. (2006). The selection strategy is based on the probability to be better than the current move, this is done by considering the standard deviation of each move. The third improvement method is inspired by the Multi-Armed Bandits problem. In the Multi-Armed Bandits problem a well known technique to optimize the exploitation and exploration trade-off is by calculating the Upper Confident Bounds for all the options, and the solution with the highest value is chosen. The logic behind this method is very simplistic; the highest valued Upper Confidence Bound has the highest potential value and therefore gives the best result. The idea of the Upper Confidence Bound is transferred to the situation with the tree structure (Kocsis and Szepesvári, 2006). The authors proposed the Upper Confidence Bounds for Trees by approaching the choice of where to select the tree as a multi-armed Bandit problem, here the value of a child node is the value estimate of the Monte Carlo Simulation over which the upper confidence bounds are calculated. Then the node with the highest confident bound is chosen. This approach naturally balances the exploration and exploitation trade-off.

2.3.2 Expansion

The second step is the expansions step, this step decides where the tree will extend. A popular expansion step is elaborated by Coulom (2006). With this method a single node is randomly added per simulation iteration, this node corresponds to the first unseen node by the tree. There are other methods, but these methods require large amounts of memory and do not improve the results of the solutions.

2.3.3 Simulation

The third step is the simulation step, in this step a roll-out algorithm is followed. Roll-out algorithms are simulations that start in the current environment state and roll-out over all the possible future states until a final state is reached. To let the algorithm converge faster a simulation strategy, or a bias, can be chosen (Chaslot et al., 2006). A simulation strategy must conform two aspects, the first aspect handles the exploration and exploitation trade off.

If the exploration or exploitation is not balanced well the model can diverge. The second aspect is regarding knowledge and search, this knowledge relates to the explicitly integrating some strategies that direct the simulation model. The search relates to the opposite, so let the simulation model search for better solutions. An example of this trade off could be integrating the standard opening moves of a chess game in a simulation model or let the simulation model search for their own. Adding knowledge often comes with high computational cost, while it can be useful to speed up convergence of the simulation model.

2.3.4 Backup

The fourth step consist of a backup phase, in this phase the trajectories found in the first two steps are backed up in a tabular format. In the works that define the different selection strategies, also different backup strategies are formulated. Besides these strategies another strategy exist. Normally the states visited in the roll-out phase are not backed up. However, a speed up method is introduced by Gelly and Silver (2007). This method gives a Rapid Action Value Evaluation of each action in state. Which means that not only the states that follow the first two steps are stored in the backup phase, but also the states in the roll out phase. This is a low variance approach that could enable a bias, the bias is counteracted with a decaying weight.

2.3.5 Retrieving best solution

After the MCTS is executed the best solution has to be gathered. This can be formalized as selecting the best child of the root node and to select this are different approaches possible. The possible options are summarized in Chaslot (2010) and are elaborated below.

- *Maximum child* - the first proposed manner to select the best child is to pick the child has obtained the highest cumulative value throughout the iterative MCTS procedure.
- *Robust child* - this method does not look at which nodes has received the highest value but rather at which child node is most visited throughout the MCTS procedure.
- *Robust max-child* - this approach selects the child node that has the highest value relatively to the highest visit count. This means that the child node with the highest percentage of successful searches is the best possible node.
- *Secure child* - this method searches for the child where the under bound is maximized. By using this approach the potentially best approach is selected.

The first two methods are straightforward but are much depending on the design choices of the MCTS because when choosing a MCTS that focuses on exploration a *robust child* approach is probably less suited because it can lead to unexpected behavior. Besides that, the other approaches create a better view of the gathered information, therefore are these approaches better suited to select the right child node.

2.4 Position of this research in the literature

The executed literature study provides the academic relevance and thereby the motivation for this project. The previous formulated solving approaches provide a good understanding of the complexities related to the TUSP. One of these complexities, the creation of robust shunting schedules, is being solved with an online approach that can be found in the reinforcement learning approaches. Reinforcement learning for the TUSP inspired an approach where the reinforcement learning agent is replaced with a set of deterministic rules. This approach is cannot be found in any of the excising solution methods for solving the Train Unit Shunting Problem. This approach is able to create robust shunting schedules but the feasibility rate lacks. Fortunately, some techniques have been reviewed that can be used to create the most optimal online algorithm and thereby improving the performance in terms of feasibility. The first technique will be an hyperparameter optimization method. Based on this literature review it appears that hyperparameter optimization techniques are often applied to optimization problems and the application to an online satisfaction problem for the TUSP is rather new and exciting. Furthermore, this literature review shows that contextual problem related information are a good addition to the hyperparameter optimization to determine the generalizability of a hyperparameter set. This project further designs techniques for searching in online scheduling environments for the Train Unit Shunting Problem. By substituting these gaps this research increases the applicability of hyperparameter optimization techniques and broadens the applicability of the online creation of shunting schedules. As a final note, this literature study is not an exhaustive literature review of the Train Unit Shunting Problem, hyperparameter optimization or search algorithms. But rather, a broad literature review to get an idea of the research field to provide a solid base for this master thesis project.

Chapter 3

Problem Description

The problem is described in the following section. First, the essential components and uncertainties of a shunting schedule are elaborated. Subsequently, the scope of the problem is described, and after that, a formal problem formulation is formulated. This chapter finishes with the design of the research.

3.1 System description

This section describes the approach that is used throughout this thesis to create a shunting plan. To create a shunting plan, a simulation model is used that is modeled as a Markov Decision Process. This simulation model is named TORS and is used to create a shunting plan by keeping track of the actions that are executed. This means that the shunting schedule is created while executing the shunting plan or formulated otherwise, the shunting plan is created online. The Markov Decision Process is formulated such that it can simulate all possible scenarios on all the available shunting yards. Therefore, the simulation model needs two components. The first component is a representation of a shunting yard and the second component is a representation of all incoming and outgoing trains, which are named scenarios. Both representations come in the form of changeable input files.

This Markov Decision Process moves from a state s to the next state s' by taking action a. To determine the right action, a set of deterministic rules is used. For simplicity, this set of deterministic rules is referred to as the *deterministic agent*. This deterministic agent keeps choosing actions until all the trains that should leave the shunting yard, have left. After all the trains have left the shunting yard without any violation, the shunting plan is feasible, and the scenario is changed to a new scenario for which a shunting schedule needs to be made. Figure 3.1 gives a visual representation of how the different components work together to create a shunting schedule. All the individual components will be explained in detail below.

3.1.1 The shunting yard location

The location input file gives a representation of the physical infrastructure. The physical infrastructure on a shunting yard consists of a finite number of tracks connected through different switches, and every track has its dedicated length. All tracks on the shunting yard have a specific function and a specific end-point. First, are the functions elaborated. The track through which the shunting yard connects to the main railway network is the gateway

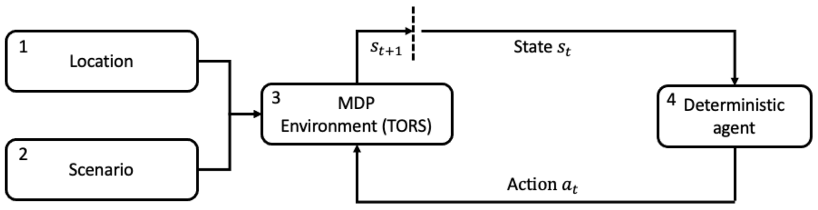


Figure 3.1: The components of the simulation environment TORS. Here (1) is the location input file, (2) is the scenario input, (3) is the Markov Decision Process environment, and (4) are the deterministic rules that control the environment.

track. This track intents for traversing to and from the shunting yard and not for long-term parking. The minor maintenance and parking can take place on dedicated service tracks and parking tracks, respectively. Most often is a shunting yard equipped with a relocation track, used to relocate trains since trains cannot overtake each other.

Each track has two end-points which are referred to as the A- and B- sides. If a track has an A- and B- side that are connected to a switch, the track is considered a *free track* and then the track can be reached through both sides. When the A- or B- side of a track is connected to a bumper and the other side is connected to switch, the track is considered a Last-In-First-Out-track or *LIFO-track*. Both the track types are depicted in Figure 3.2, and both track types need a completely different approach regarding the scheduling of the activities on the shunting yard. Therefore, two main shunting yard structures can be distinguished, the shuffleboard structure and the carousel structure. The shuffleboard structured shunting yards consist mainly of LIFO-tracks which require an advanced relation strategy, and the carousel structured shunting yard has many free tracks which enable a carousel-like movement.

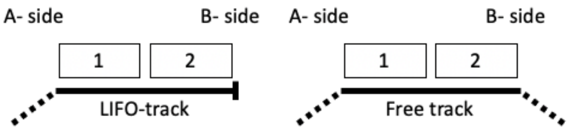


Figure 3.2: Difference between a LIFO-track (left) and free track (right). The referred tracks are depicted with the solid lines with the parked trains on top of them, the dotted lines are the adjacent tracks. The numbers of the trains corresponds to the number of departure and requires a different parking strategy.

3.1.2 The shunting scenario

The trains that are used to create a shunting schedule are specified in the shunting scenario. A shunting scenario consists of a specification of the trains that will arrive at a shunting yard and the trains that will depart at the shunting yard. The trains considered in these shunting schedules are self-propelling vehicles that can drive in both directions. A train can consist of three train units, but throughout this thesis, a maximum of two train units is considered to control complexity. The trains that will be used during the thesis are the *Verlengd InterRegio Materieel* (VIRM), *Sprinter Nieuwe Generatie* (SNG) and the *Sprinter LightTrain* (SLT). More specifically, the VIRM-4, VIRM-6, SNG-3, SNG-4, SLT-4, and SLT-6, where the number refers to the number of carriages of each train unit. To create a smooth flow of trains from and to the shunting yard, the schedule is made based on the planning of the main train network.

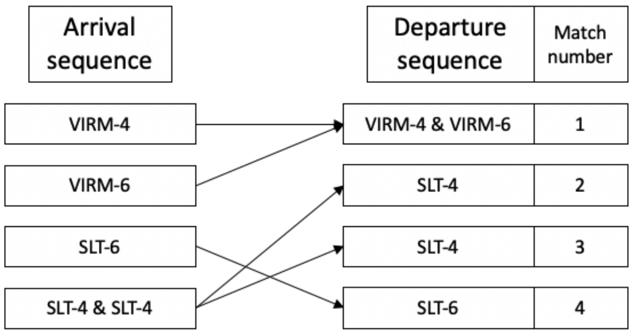


Figure 3.3: representation of the arrival sequence and departure sequence with the matched numbers accordingly.

Therefore, the incoming trains arrive at predefined times and the departing trains have to leave at predefined times. Both the predefined arrival and departure times are formulated in the scenarios. Assigning different incoming train units to an outgoing train is referred to as the matching problem. This matching problem is not considered throughout the master thesis, and therefore is each train in the scenario file equipped with a predefined match number. This number represents the allocation of an incoming train unit to a specific outgoing train, where the number is the numerical order of leaving. To create the correct departing configuration, arriving trains must be split and recombined to form the outgoing trains. An example of an arrival and departure sequence with the matching numbers is given in Figure 3.3. As can be seen in the figure, if two trains have to be combined to form a departing train, they obtain the same match number. Besides that, every train unit has its own service time specified in the scenario file. The service time variable must be zero before leaving the shunting yard and executing the service can only happen at predefined service tracks. Some trains enter the shunting yard and do not need any service. With these trains, the service time is already zero in the scenario file.

3.1.3 The simulation environment

The location and scenario files are input variables for the simulation environment. This event scheduling simulation environment is named the Train Maintenance and Shunting simulator (TORS[1]). This Markov Decision Process is based on a discrete event scheduling technique. This means that actions are taken in response to events, better known as triggers. The Markov Decision Process stores these triggers and each trigger has its own time of execution. The environment executes all triggers incrementally, based on the specified time. When a trigger is executed, the trigger is removed from the trigger list. The triggers are added to the environment through two methods. Firstly, by the events specified in the scenario file that have to happen at strict predefined times, these are the arrival and departure events. The second method to add triggers to the environment is in a reaction to other previously executed triggers. When for example, a train decides to start a service action, that action creates a 'service-ended-trigger'. On this trigger, the train is done with the service, and that train, or another train, can execute an action. Because the movement of a specific train has much influence on the other trains, multiple other triggers are added before and after

[1]This abbreviation originates from the dutch translation.

the existing triggers to let other trains move. The deterministic agent uses these triggers to choose actions that will be executed on the environment. For all trains in the simulation model, similar actions are available. The actions that are available per train are explained briefly because most of the actions are self-explanatory.

- *Departure* - The departure action lets a train leave the shunting yard. With this action, the train is removed from the simulation model.
- *Split* - The split action splits a train in the simulation model, consisting of two train units, into two trains.
- *Service* - This action activates the service. The service actions can be executed on multiple dedicated tracks. This action can be considered a move action but then to the dedicated service tracks. All the possible end-tracks are considered individual actions.
- *Connect* - The connect action is the opposite of the split action. Two trains merge into one train. The composition of which train units are used is specified in the scenario file.
- *Setback* - The setback action sets the train in neutral gear such that the train can drive both directions in future states. A setback action is required before a train can switch from driving direction.
- *Wait* - The wait action lets the train do noting and let the possibility for other trains to execute actions without any interruption.
- *Move* - The move action, moves the train from a starting track to an end-track. Here, all the end-tracks are specified as a single action.

Besides that, there is an *arrival* action. This action is not chosen by the deterministic agent but is executed when the trigger occurs. Then no other actions can be executed. Logically, the action selection procedure starts after a train has arrived at the shunting yard and ends after the train has departed from the shunting yard.

After an action is selected, that action will be executed on the environment. The execution may lead to a violation of the environmental constraints. When one of these constraints is violated, the shunting plan becomes infeasible. Then, the generation of the shunting schedule stops and the environment starts over with another scenario. These violations are elaborated below.

- *Arrival track reserved* - This violation occurs when the gateway track, on which the trains arrive, is fully occupied and the action is an arrival action that ensures a new arrival.
- *Train cannot leave* - This violation occurs when a departure action is chosen and the right train was not on the gateway track. More specifically, the train with matched number one is not presented on the gateway track.
- *Did not depart* - This violation occurs when a departure action must be chosen, the wrong action is chosen, or the train was blocked by another train on the gateway track.
- *Composition not present* - The fourth violation is rather straightforward, this violation occurs when a departure action is chosen and the train is not in the right composition.
- *No actions left* - The last violation occurs when there are no actions to choose from for the deterministic agent.

3.1.4 The deterministic agent

The deterministic agent is the component that is responsible for the action selection. Before this deterministic agent selects an action, it filters the infeasible actions. An action is infeasible if the action cannot be executed in the environment due to action-related reasons. More specifically, a departure action is infeasible when the dedicated train is not on the departure track and the split action is infeasible when the train only consists of one train unit. Movement or service actions become infeasible if the traversing track is not available or the end track is fully occupied. Besides that, the service action becomes infeasible when the train does not need any service. The connect action becomes infeasible when there are not two trains next to each other that need to be combined or if a train consist already of two train units. All other actions are always feasible.

After the feasible actions per train are determined, a set of deterministic rules is used to determine the best action. The simulation model goes through each of the actions per train and assigns a priority value to each by evaluating different deterministic rules. If a deterministic rule is true, the priority value is assigned, and after all actions of all the trains are valued, the action with the highest priority value is selected.

The deterministic rules can be divided into two categories. The first category assigns a priority value based on the availability of an action. If an action is considered feasible, a priority value is assigned. The actions for which such an approach is used are the *Departure, Split, Connect, Setback* and *Wait* actions. The specific values differ per location, but the priority values of the shunting yard Kleine Binckhorst are formulated to give an impression (Table 3.1). These priority values are specified through experts' domain knowledge by estimating which action is more important. The actions in the other category are movement-

Table 3.1: Priority values specified for the shunting yard Kleine Binckhorst

Action	Depart	Split	Connect	Setback	Wait
Reward	400	380	280	200	100

related and require a different approach with more rules. These *Service* and *Move* actions are different because every move action can contain multiple different viable subactions. Formulated otherwise, for every reachable end track, a move action is created and if these end tracks are service tracks, it becomes a service action. This creates lots of complexity since there needs to be a distinction between these different end tracks. Currently, the distinction between different actions is made by formulating more deterministic rules that consider the different functions of tracks. If a subaction satisfies a deterministic rule, a priority value is assigned to that subaction. The functions of the tracks are elaborated in section 3.1.1. Besides these functions, some extra constraints are introduced that prioritize specific actions more than other actions. These constraints are based on the next event, the service time and the matched number of a train. A specification of these movement actions for the location Kleine Binckhorst are formulated in Table 3.2. In this example the are the parking tracks divided in {Parking A} and {Parking B} for the tracks {2,3,4,5} and {6,7,8,9} respectively. The assigned priority values are specified through experts' domain knowledge by estimating which action is more important.

The formulated deterministic rules and constraints are used to distinguish the importance between the functional groups of tracks. However, each function group can consist of multiple tracks. Therefore, a small *extra reward* is added to each assigned value. These extra rewards

Table 3.2: Priority values specified for the movement heuristic on shunting yard Kleine Binckhorst. The function of a track is represented between the brackets where the parking tracks are divided in {Parking A} and {Parking B} for the tracks {2,3,4,5} and {6,7,8,9} respectively. Besides that, extra constraints and extra rewards rewards are allocated.

#	**Action**	**Extra constraint(s)**	**Assigned value**
1	From any to {Gateway}	Match number = 1, next event = departure event	370 + extra reward
2	From {Service} or {Relocation} to {Parking} or {Relocation}	Match number = 1, service time = 0	220 + extra reward
3	From {Gateway} to {Parking B}	Service time > 0	239 + extra reward
4	From {Gateway} to {Service}	Service time > 0	240 + extra reward
5	From {Gateway} to {Parking B}	Service time = 0	228 + extra reward
6	From {Gateway} to {Relocation}	Service time = 0	230 + extra reward
7	From {Gateway} to {Parking A}	Match number ≠ 1, next event = departure event, service time = 0	228 + extra reward
8	From {Gateway} to {Parking A}	Match number ≠ 1, Service time > 0	201 + extra reward
9	From {Parking B} to {Service}	Service time > 0	105 + extra reward
10	From {Parking A} to {Relocation}	Service time = 0	101 + extra reward
11	From {Service} to {Relocation}	Service time = 0	106 + extra reward
12	From {Service} to {Parking}	Service time = 0	102 + extra reward
13	From {Relocation} to {Parking}	NA	107 + extra reward

are small priority values that make a difference between multiple tracks within a similar function group, based on the trains already on these tracks. This is done by comparing the matched number of the train that is on a specific track with the matched number of the train that analyses the action. When the train on a specific track has a lower match number than the train that wants to execute the action, a small priority value is added based on the difference of the matched numbers. Here, the highest extra reward is assigned when the difference in match numbers is the closest to zero. This is done because the trains on a track determine which specific track would be the best end track considering the departure sequence. For the location Kleine Binckhorst are these extra rewards in the range from 0 till 3.

After all the trains' actions are valued, the algorithm selects the action with the highest priority value. This action is executed in the simulation environment. However, in some cases, the highest valued action cannot be determined, which is caused by two exceptions. The first exception occurs when there are multiple equal highest valued actions from different trains. Then, the train with the lowest matched number is given priority since a lower match number means an earlier departure, and therefore, this action will be more important than the other actions. The second exception occurs when multiple actions have obtained an equal and highest value within the same train. Such a situation occurs when the extra reward cannot distinguish the priority between different tracks, mostly when the tracks are empty. Then the importance of an action cannot be determined since that depends on other trains on that track. In such cases, an arbitrary action is chosen to be executed in the simulation environment. A summary of the action selection procedure of the deterministic agent is depicted (Figure 3.4).

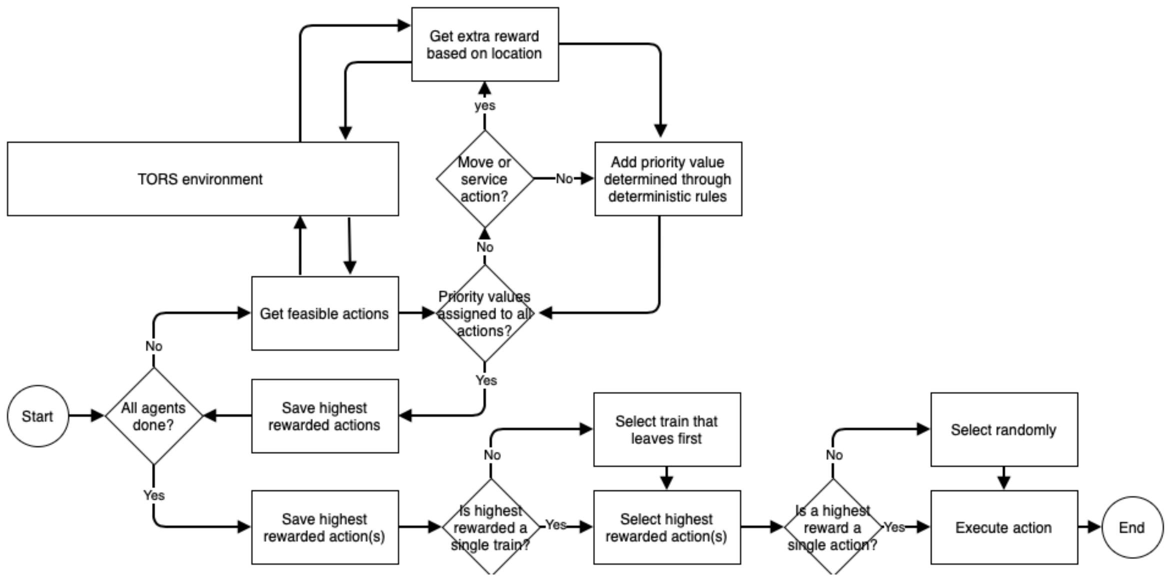

Get extra reward based on location
yes
TORS environment
Move or service action?
No
Add priority value determined through deterministic rules
No
Get feasible actions
Priority values assigned to all actions?
No
Yes
Start
All agents done?
Save highest rewarded actions
Select train that leaves first
Select randomly
No
No
Yes
Save highest rewarded action(s)
Is highest rewarded a single train?
Yes
Select highest rewarded action(s)
Is a highest reward a single action?
Yes
Execute action
End

Figure 3.4: Process flow of the action selection procedure of the deterministic agent.

3.1.5 Suboptimal feasibility

The deterministic agent can select sufficient actions sequentially and thereby making feasible shunting schedules. However, the number of feasible solutions is still not as desired. This subsection identifies improvement areas in creating shunting schedules that will be addressed in the remainder of this thesis.

Within the deterministic agent, the deterministic rules assign a priority value to an action. The assigned value determines the importance of an action compared to all other actions because the highest valued action will be chosen. The values specified in Table 3.1 show that for the location Kleine Binckhorst the departure action is the most important action and the wait action is the least important action, since these actions have the lowest and highest values, respectively. When these priority values are changed, take for example that the values of these two actions are switched, which means that the departure action becomes the least important action and the wait action becomes the most important action. Then, the dynamics of the deterministic agent will change to a situation where it will always choose the wait action. Such a change is not desirable, but it gives a clear example that changing the priority values will change the dynamics of the deterministic agent.

The assigned priority values can be considered as hyperparameters. Because, these priority values determine the dynamics of the shunting plan generation while remaining fixed throughout the process of executing the actual schedule. Currently, the assigned priority values are determined through domain knowledge by estimating which action is more important, these are not determined analytically. Besides that, the hyperparameter configurations are currently formulated per location. This means that all the scenarios are solved with similar generalized hyperparameters. It is thought that this is suboptimal since problem-related contextual information might distinguish the importance of different actions. For example, a scenario with 15 train units is currently solved with similar hyperparameters as a scenario with 17 train units. However, a scenario with 15 train units does not require combination actions and with 17 trains, these combination actions are necessary. The number of splits and combinations are equal within all scenarios since we tried to make the scenarios as homogeneous as possible.

Ideally, all possible hyperparameter configurations are checked because in potential can optimizing the hyperparameters lead to more solved instances. However, this is suboptimal since a few hyperparameters lead to many possible configurations, which becomes hard to evaluate when considering the computation times. To overcome the exhaustive search to the right hyperparameter configuration, a hyperparameter optimization technique will be used that will consider problem related contextual information.

Another improvement area can be noticed in the second exception within the selection of movement actions of the deterministic agent. This second exception occurs when multiple actions have obtained an equal and highest value within the same train. Such a situation occurs when the earlier formulated extra reward cannot distinguish between different tracks, mostly when the tracks are empty. Then the importance of an action cannot be determined since that depends on other trains on that track. In such cases, an arbitrary action is chosen to be executed in the simulation environment. However, this arbitrary action is suboptimal due to the lengths of the tracks in combination with the arrival and departure sequences. This will be elaborated through an example that is depicted in Figure 3.5. As can be seen in this figure, there is a shunting yard with two empty parking tracks of different lengths. The first track has enough space for a single train, and the second track has enough length for two trains. When three trains need to be parked on these tracks, an optimal configuration can be made based on the layout of the tracks and the arrival and departure sequences. Assuming that all the trains must be able to move to the gateway track unobstructed. In this figure are multiple arrival sequences given with the numerical order of leaving where the lowest number has to move to the gateway track first. Putting an arbitrary action in a shunting plan is inefficient

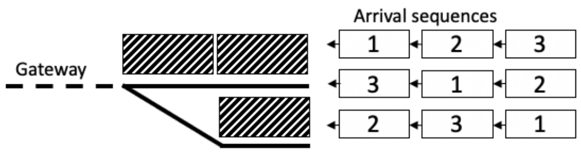
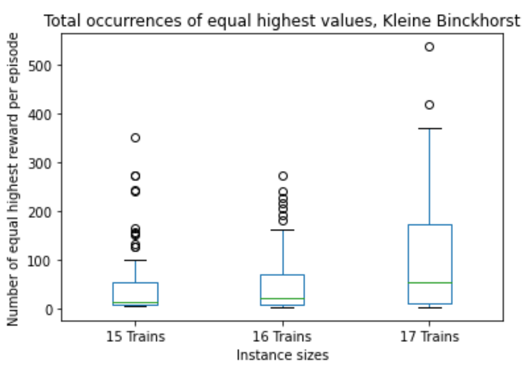


Figure 3.5: A graphical proof that an arbitrarly choice is suboptimal when there are multiple movement actions that have obtained an equal and highest value from the same train. When there a three parking position and three trains that arrive the optimal parking configuration can only be determined by considering the arrival and departure sequence.

since an action could cause the shunting plan to become infeasible. To give some insights about the arbitrary actions, an initial measurement is executed. This initial measurement counts the number of arbitrary actions that occur in a scenario and for this measurement 100 scenarios are analyzed. This is measured for two location types, a carousel-like location named Kleine Binckhorst and an Idealize Location that is named Idealized Location for simplicity. The results of the initial analysis are visible where the left plot represents the Kleine Binckhorst and the right plot the Idealized Location (Figure 3.6). The results for the location Kleine Binckhorst show a higher median of arbitrary actions if compared with the Idealized Location. Besides that, there is a noticeable difference in the deviation of the measurements. More specifically, the higher the number of trains, the greater the deviation in the number of arbitrarily moves, especially for the location Kleine Binckhorst. This means that Kleine Binckhorst has the most potential to benefit from improving the random moves, especially the scenarios with 17 train units. To overcome this suboptimal arbitrarily action



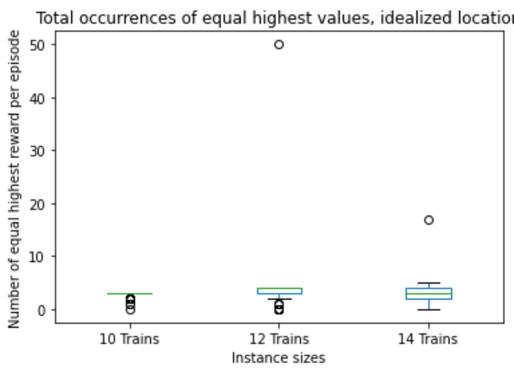


Figure 3.6: Results of the initial measurement on the shunting yard Kleine Binckhorst.

selection, a search algorithm will be introduced. This search algorithm tries to extend the deterministic agent to handle the changing arrival and departure sequences. Formulated otherwise, a search algorithm will be used to handle the stochasticity found in the specified arrivals and departures of the analyzed scenarios.

3.2 Scope of the problem

The scope of the problem can be defined from the shunting schedule system description and the formulated inefficiencies. This research will focus on improving the shunting schedule generation by optimizing the hyperparameter configuration and handling the stochasticity found in the scenarios. The following aspect will not be included in this research:

An explicit coverage of the routing - the routing of trains is necessary to enable the movements of trains on the shunting yard. This is formalized as the routing problem. The routing is implied through the event scheduling simulation environment but not explicitly analyzed or adjusted. Always the shortest possible routing option is chosen when traversing from a start track to an end track.

3.3 Train Unit Shunting Problem through a deterministic agent

The main challenge is to increase the number of online created feasible shunting plans by optimizing the deterministic agent. The problem of creating a shunting schedule online can be formalized as follows:
Given:

- *A shunting yard location with tracks*
- *and a scenario file with specified arrival and departure events*

The deterministic agent needs to:

- *Sequentially decide, for all the trains on the shunting yard, which action to take*

Such that:

- *All trains can arrive and leave the shunting yard unobstructed*

- *and no violations occur by selecting the wrong actions.*

This challenge can best be formulated as a Constraint Satisfaction Problem. The main difference between a Constraint Satisfaction Problem and a typical optimization problem is that Constraint Satisfaction Problem tries to generate a feasible solution with the given constraints instead of an optimal solution regarding some objective function. Currently, a shunting plan is called infeasible when the deterministic agent chooses an action that triggers a violation, and a shunting plan is called feasible if all trains can leave the shunting yard unobstructed. Therefore, the specified violations can be considered the constraints that have to be satisfied. The feasibility of each scenario will be represented through a boolean value and the boolean values of multiple analyzed scenarios will form the objective function. Throughout this thesis, we will focus on satisfying and improving the number of feasible solutions by creating a framework of two different approaches. The first approach focuses on optimizing the hyperparameter configuration of the deterministic agent, and the second approach will improve the deterministic movement rules by incorporating a search strategy that will handle the stochasticity.

3.3.1 Hyperparameter optimization

To optimize the hyperparameters, a model-based optimization method will be used. Model-based optimization methods try to construct a mapping between a set of hyperparameters and the resulting number of feasible solutions. This mapping will eventually be used to select the best hyperparameter configuration. Sequential Model-Based Optimization (SMBO) puts the mapping and selecting the optimal hyperparameter set in an iterative procedure to find the optima of the objective function.

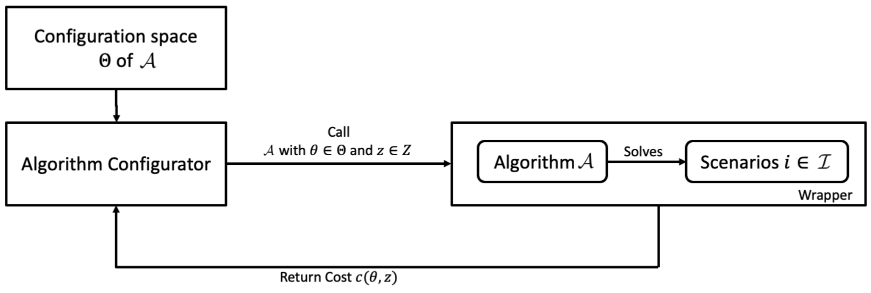

Configuration space
Θ of $\mathcal{A}$
Algorithm Configurator
Call
$\mathcal{A}$ with $\theta \in \Theta$ and $z \in Z$
Algorithm $\mathcal{A}$
Solves
Scenarios $i \in \mathcal{I}$
Wrapper
Return Cost $c(\theta, z)$

Figure 3.7: General optimization framework, adjusted for the actual problem (retrieved and modified from Eggensperger et al. (2019)).

A graphical representation of such an iterative procedure is given (Figure 3.7). Our work uses the iterative procedure as follows; a Bayesian Optimization (BO) technique will be used as an algorithm configurator. The BO selects a hyperparameter set (θ_n) in the configuration space by using its acquisition function. This hyperparameter set is used in our TORS environment with the deterministic planner to solve multiple scenarios. Eventually, the objective of a feasibility rate (c_n) and an error term (ϵ_n) is returned to the BO. BO creates the mapping between the hyperparameters and all historically obtained results. This mapping is the surrogate model and the BO uses this surrogate model within an acquisition function to select the best hyperparameter set to test next. With every iteration, the mapping improves until a stopping criterion is met, which will end the optimization procedure. A stopping criteria can consist of a predefined number of iterations. This is all while considering problem-related contextual information (z_n). The error term, on the obtained result, consists of independently distributed values across the observations. The pseudo-code of a basic BO formulation is formulated algorithm 1.

Algorithm 1: Bayesian Optimization Algorithm

```
Input: Algorithm $\mathcal{A}$, acquisition function $a$, surrogate model $f$, iterations $N$, historical queried
  configurations $\mathcal{D}$, context $z$, configuration space $\Theta$
for $n = 1, 2, \ldots, N$ do
    Select; $\theta_n = \underset{\theta \in \Theta}{arg\,max}\ a(f)$
    Calculate; $c(\theta_n, z_n, \epsilon_n) = \mathcal{A}(\theta_n, z_n)$
    Update; $\mathcal{D}_n$ with $(\theta_n, z_n, c_n)$
    Calculate; $f$ based on $\mathcal{D}_n$
end
```

3.3.2 Handling of stochastic variables

The second part of the framework incorporates a search strategy that handles the stochasticity that occurs when analyzing different problem instances. These stochastic variables are the arrival and departure sequences that need to be considered when the deterministic rules cannot distinguish different tracks with similar functions. The search strategy that will be used is the Monte Carlo Tree Search (MCTS). The MCTS follows a tree structure where all the nodes represent the states and the arcs represent the movement actions. Every MCTS iteration adds a new trajectory and the success rate of that trajectory to the tree. Such a trajectory represents the sequential selection of movement actions and after many iterations, Monte Carlo estimates of the optimal movement become apparent. These Monte Carlo estimates will be used to determine the best action for a particular train in a deterministic manner while considering the current state of the environment. Moreover, these Monte Carlo estimates are a deterministic manner to summarize the stochastic arrival and departure sequences.

The MCTS consist of four steps which are executed in an iterative manner. These steps are repeated until the computational resources are exhausted. When the computational resources are exhausted, the Monte Carlo estimates can be derived. Then, the environment moves to a new state and the procedure starts again. The four steps of the MCTS are formulated below.

- *Selection* - From the current simulation state, also formulated as the starting node, a tree policy is followed based on the earlier obtained MCTS values to traverse to a new state.
- *Expansion* - On arbitrarily iterations, the tree is expanded to make the solution space of the search tree larger.
- *Simulation* - Through the previous two actions, a state is selected, and from this state onward, the policy used within the MCTS is changed from a tree based policy to a random policy. This means that the remaining solution space is searched with a roll-out algorithm. This is done until a terminate state is reached.
- *Backup* - The actions taken in the first two steps are then backed up. This is done by keeping track of the number of passes and the success rate of each pass. This means that the states visited during the simulation step are not backed up.

3.4 Research design

In this section, the setup of the research will be elaborated. Throughout this thesis, the problems formulated in the past chapter will be solved. The next chapter, chapter 4, will elaborate on the application of the solution methods. Eventually, in chapter 5 the experimental data is generated. This is done for both the challenges simultaneously. The results for all the experiments are discussed in chapter 6 and in that chapter, some additional analyses will be done. Finally, all findings of the research are concluded in chapter 7.

Chapter 4

Solution Methods

This chapter describes the proposed solution framework. The first part of the solution framework consists of the details of the hyperparameter optimization through Bayesian Optimization. The second part of the solution framework focuses on implementing a search strategy through Monte Carlo Tree Search.

4.1 Bayesian Optimization for hyperparameter optimization

This section explains how the hyperparameter optimization method will integrate with the TORS environment and the deterministic agent. After that, the surrogate modeling technique and acquisition function of the Bayesian Optimization will be elaborated.

4.1.1 Adjustments to the TORS environment

The implementation of the hyperparameter optimization will be done with an optimization framework. An example of a general optimization framework is given in Figure 3.7. As seen in that figure, the algorithm configurator is placed as a wrapper over the algorithm for which the hyperparameters need to be configured. A similar approach will be used throughout this work. We will create a wrapper that will be placed over the TORS environment with the deterministic agent. Our wrapper will be formulated such that the interaction between the wrapper and the TORS environment with the deterministic agent only consists of the hyperparameters' values and the obtained results.

In total, eight continuous hyperparameters will be optimized through this wrapper. Each hyperparameter equals the priority value of an action in the TORS environment and the movement action will be divided into two hyperparameters. This means that the following actions will be considered throughout the hyperparameter optimization: departure, split, connect, setback, wait, service and the movement actions. The movement actions are divided in two categories, namely, 'move entering' and 'move internal'. These are only two categories to control the dimensionality of the model and thereby cope with the curse of dimensionality that would occur when all the movement actions are added individually. Besides that, the underlying movement strategy is currently not considered as a bottleneck. With this deviation of the movement actions, the entering movements are concerned with the movements that have the gateway track as start track and the internal movements are all other possible movements. The difference between these two categories is made because there is a visible distinction in the defined priority values (see Table 3.2 for the location Kleine Binckhorst). Besides that, the entering movements are more important than the internal movements since a violation is triggered if the gateway track is fully occupied.

To combine the hyperparameter optimization with the initially defined movement strategy, a combined priority value is created. This priority value consist of a part that is determined through the hyperparameter and a part that has the value it is originally designed with. To give an example for this; if a train on the location Kleine Binckhorst wants to move from the gateway track to the service track, a priority value of 240 is originally given. This priory value is split into 200 and 40 where the 200 can be modified through the hyperparameter optimization, and the 40 is remained fixed to keep the original movement strategy intact. Logically, the part determined through the hyperparameter optimization is equal in size for all movements throughout the move entering and move internal category. Formulated specifically for the location Kleine Binckhorst (Table 3.2), for the move entering actions, a value of 200 is subtracted, which will be determined through the hyperparameter optimization. With the internal movements, this value becomes 100.

Besides that, the results have to be gathered. Usually, the TORS environment with the deterministic agent creates a shunting plan as output. This output is changed to a representation of the feasibility of the analyzed scenarios. The feasibility of a set of scenarios is determined through a summation of the binary variables if a shunting schedule is feasible or not. This value will be returned to the Bayesian Optimization wrapper.

Finally, an improvement is executed, which changes the structure of the optimization wrapper. This is because the TORS environment with the deterministic agent is designed to use only a single logical core of the computer while executing. This creates the opportunity to run different TORS environments, each with a deterministic agent, simultaneously. In a practical sense, this means that every TORS environment must be allocated to a predefined set of problem instances. This means that the process is split into multiple subprocesses before the simulation starts. When all the subprocesses are done simulating, the processes join back to a single core to calculate the Bayesian Optimization. When processes are joined back together, the results are aggregated too. This way of working means that it could happen that a logical core must wait on another core, but roughly this decreases the computation time with a factor m, where m is equal to the number of logical cores of a machine. The machines used for the Bayesian Optimization have eight logical cores, which means that the number of scenarios of the experiments is a multiple of eight.

The changes in the wrapper that is necessary to efficiently combine the Bayesian Optimization and the TORS environment with the deterministic agent are graphically represented (Figure 4.1).

4.1.2 Surrogate model and acquisition function

The surrogate or response surface model tries to map the selected hyperparameters and contextual information and the objective function while considering the noise. Different methods can be used to create the response surface model. The best-known approaches are Gaussian Processes (Rasmussen, 2003), Sequential Model-based Algorithm Configuration (Hutter et al., 2010), and Tree-Parzen Estimator (Bergstra et al., 2011). The difference between these approaches is that Gaussian Processes (GP) uses Gaussian regression models to create the response surface model where the Sequential Model-based Algorithm Configuration (SMAC) and Tree-Parzen Estimator (TPE) are using structures based on decision trees. Therefore,

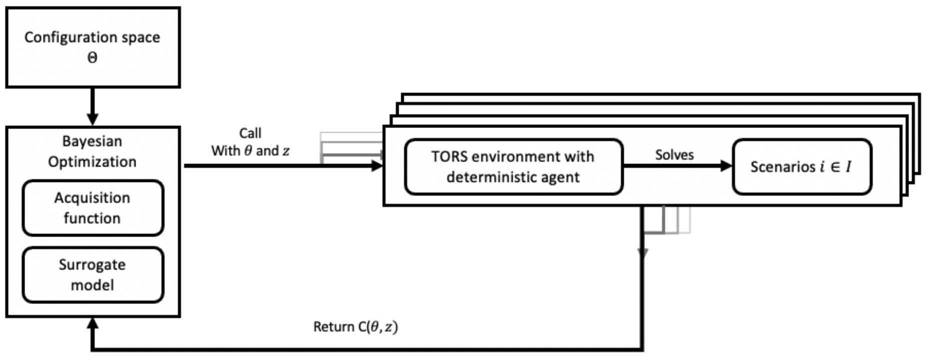


Figure 4.1: Bayesian Optimization wrapper used throughout this thesis.

the SMAC and TPE approaches are better suited when considering a more general search space consisting of categorical and discrete values. Subsequently, these approaches require more evaluation iterations to give accurate predictions. When using GP with categorical or discrete values, modeling tricks must transform the values into the correct format. The time complexity of all approaches differs a lot. GP consists of matrix multiplications which have a time complexity of $\mathcal{O}(n^3)$ where n represents the dimension of the $n \times n$ matrix (Strassen, 1969). SMAC and TPE use decision trees and to give an indication of the time complexity, binary decision trees are used. The time complexity of binary decision trees are dependent on the number of nodes, when each node requires $\mathcal{O}(1)$ to calculate, the time complexity of the decision tree is bounded by the depth of the tree. Then the decision tree has a time complexity of $\mathcal{O}(n)$ where n represent the depth of the decision tree (Buhrman and De Wolf, 2002). Therefore, the preference of the computational complexity is in favor of the decision trees; however, the number of necessary samples to give a solid prediction is lower with the GP approach. This is because the SMAC and TPE require multiple observations with the same hyperparameter configuration. Besides that, all the hyperparameters used throughout this thesis are continuous variables that does not prefer a method over another.

In this research is chosen to apply the Gaussian Processes in the Bayesian Optimization framework to create the surrogate model. This is because the TORS environment with the deterministic agent is an expensive algorithm and therefore an approach is preferred that requires the least number of samples from this expensive algorithm. Formulated otherwise, the surrogate modeling technique is used that is as efficient as possible with the obtained samples. Besides that, the underlying time complexity of the surrogate model does not become a bottleneck. These computation times are negligible compared to the computation times that arise when using the TORS environment with the deterministic agent more often. Besides that, the GP work well when considering only continuous variables.

GP is a stochastic process, often referred to as a collection of random variables, such that every joint distribution of those random variables has a multivariate Gaussian distribution, meaning that every linear relationship between those variables is normally distributed. Another advantage of the GP is that conjugation of prior GP distributions still results in a GP distribution. The similarities between multiple sets of hyperparameters are denoted through a single Gaussian Process with $GP(\mu, k)$ where, $\mu : d$ is the mean and $k : d \times d$ is the kernel or covariance-function, $\mu, k \in \mathbb{R}$. These single GP formulations can be extended to multivariate Gaussian Process formulations. This requires an extension where the covariance is transformed to a covariance matrix. The proof of this is elaborated in Girolami (2011).

Besides the surrogate model, an acquisition function is needed that indicates the best following hyperparameter configuration to sample. Like formulated in chapter 2, there are different acquisition functions that all have their properties. As a reminder, often used acquisition functions are the Probability Of Improvement (Kushner, 1964), Expected Improvement (Močkus, 1975), and Gaussian Process Upper Confidence Bound (Srinivas et al., 2009). Probability Of Improvement (POI) returns the hyperparameters based on the point where the most improvement can be achieved. This can result in some strange behavior where the best point is selected based on the current minimum, which is unrelated to the size of the improvement. Subsequently, Expected Improvement (EI) is a modification on POI that considers the expected improvement in these cases. The last option is the Gaussian Process Upper Confidence Bound (GP-UCB) that searches the highest valued upper confidence bound and returns that hyperparameter set. With GP-UCB, the position is returned with the most potential to be the highest evaluated set of hyperparameters. A more elaborated explanation on these methods can be found in chapter 2.

$$a(\theta_n, \mathcal{D}_n) = \mu_{n-1}(\theta) + \sqrt{\beta_n}\sigma_{n-1}(\theta) \tag{4.1}$$

$$\theta_{n+1} = \underset{\theta \in \Theta}{arg\,max}\ a(\theta_n, \mathcal{D}_n) \tag{4.2}$$

For this research, the chosen acquisition function is the GP-UCB. This is done with Equation 4.1 where the first term μ_{n-1} is responsible for the exploitation of existing areas. The second term $\sigma_n(\theta)$ represents the standard deviation at point θ. This term is responsible for the exploration of new areas in the search space. The importance of both parts can be determined with the constant β. When the upper bound is determined, the Equation 4.2 is used to determine the highest value. The highest value is then transformed to a set of hyperparameters that will be sampled next.

The actual implementation of the wrapper that contains the Bayesian Optimization with the surrogate modeling and the acquisition function is visible in the pseudo-code below.

Algorithm 2: Implementation of the Bayesian Optimization framework

```
Input: Context $z$, number BO iterations $N$, number of logical processes $LP$, Batch size $B$, set with
  problem instances $I$, TORS environment with deterministic agent $\mathcal{A}$.
Output: $c(\theta_n, z_n, \epsilon_n)$ after testing, $c(\theta_n, z_n, \epsilon_n)$ from $\mathcal{D}_N$
Initialize surrogate model $f$
Initialize queried training configurations $\mathcal{D}_0$
Create training set and test data by excluding a random sample of size $B$ from instances $I$
for $n = 1, 2, \ldots, N$ do
    if $n < 10$ then
        Select hyperparameters $\theta_n \in \Theta$ randomly
    else
        Calculate $f$ based on $\mathcal{D}_{n-1}$
        Select hyperparameter set $\theta_n$ from acquisition function with $\underset{\theta \in \Theta}{arg\,max}\ a(\theta_{n-1}, \mathcal{D}_{n-1})$
    end
    Select training batch with size $B$ randomly from training set
    Create subprocesses equal to number of $LP$ and divide training data over $LP$
    Calculate result $c(\theta_n, z_n, \epsilon_n)$ for training data with the TORS environment with the
      deterministic agent $\mathcal{A}(\theta_n, z_n)$
    Merge subprocesses and $c(\theta_n, z_n, \epsilon_n)$
    Update queried configurations $\mathcal{D}_n$ with $(\theta_n, z_n, c_n)$
end
Select hyperparameter set $\theta_n$ with the highest $c(\theta_n, z_n, \epsilon_n)$ from $\mathcal{D}_N$
Create subprocesses equal to number of $LP$ and divide test data over $LP$
Calculate result $c(\theta_n, z_n, \epsilon_n)$ for test data with the TORS environment with the deterministic agent
  $\mathcal{A}(\theta_n, z_n)$
Merge subprocesses and $c(\theta_n, z_n, \epsilon_n)$
```

4.2 Monte Carlo Tree Search for handling stochastic variables

This section describes how the Monte Carlo Tree Search will be integrated with the deterministic agent. After that, some aspects are formulated that will increase the computational efficiency.

4.2.1 Adjustments to the TORS environment

The TORS environment is an event scheduling simulation environment that moves from state s to state s' by taking action a. Every state in this simulation environment represents a summary of all the available information in a shunting yard. However, the problem solved with the Monte Carlo Tree Search only relates to movement actions. This means that there is redundant information in the state space representation, making the MCTS inefficient if used. Therefore a *MCTS state space* is formulated that only considers tracks where the stochasticity occurs. For the location Kleine Binckhorst this is on the service and parking tracks. The stochasticity per type of track will be handled through individual models with their own MCTS state space. Therefore, the explanation of the solution methods will only consider the parking tracks for the location Kleine Binckhorst, but the service tracks and the stochasticity on other shunting yard locations will be approached similarly. The MCTS states are formulated by dividing the tracks into dedicated parking spots, which form an MCTS state. The parking spots are based on the smallest train that is available in the scenario. This means that when there are four tracks of 180 meters, and the smallest train is 60 meters, there are 12 parking positions. Every time a train is parked on these four tracks, the state changes from s to s'. The actions that are responsible for the parking of trains are defined as the tracks numbers. When an action is chosen, the train is parked behind the trains on that specific track and if the track is empty the train is forwarded to the beginning of the track.

These parking positions combined with the trains that arrive at a shunting yard are transformed to a tabular format. This matrix will be used to store the data which will be gathered in the backup phase. The states and actions are used to create a search tree of simulated trajectories. A trajectory is created by parking trains on tracks sequentially until all trains in the arrival sequence are parked. After all the trains in an arrival sequence are parked on the tracks is checked if all trains can leave the shunting yard unobstructed, according to the given departure sequence. The implementation holds two similar matrices, both matrices count which trains are on what position in the last state of the trajectory. However, one matrix counts all the solutions and one matrix counts only the feasible solutions. In this manner, the feasibility of certain parking positions for specific trains can be determined.

The sequence in which trains arrive at the shunting yard is not how the trains are parked on the parking tracks. This is because each train can overtake other trains while moving towards the parking tracks. This happens for example when a train needs service, then that train can easily be overtaken by another train that does not need any service. Therefore, the arrival sequence is randomized with every MCTS iteration. This different arrival sequence will eventually create all possible parking configurations. The MCTS iterations result in two filled matrices that, when divided, show percentages of logically distributed parking possibilities because it is impossible to place the train which has to leave first behind another train.

To use the MCTS properly, a communication mechanism will be placed between the MCTS state space and the TORS environment with the deterministic agent such that the MCTS can always provide the most actual information. This is realized by creating a reactive MCTS that starts when the tracks are empty, then MCTS iterations fill the matrices. After the matrices are filled, the deterministic agent continues assigning priority values to the actions and if the deterministic agent cannot assign a priority value deliberated, additional information is retrieved from the matrices. When the deterministic agent has assigned a priority value to all the actions. The action with the highest priority value will be executed on the TORS environment which lets the TORS environment move to the next state. Then the matrices are emptied and the procedure starts again. This new iteration starts with transferring the new occupation of the parking tracks to the MCTS state space and the matrices are renewed with simulation results of the adjusted arrival sequence. This iterative procedure keeps supporting the deterministic agent until parked trains start to leave the shunting yard. Then too many changes are happening in the parking tracks for which the MCTS continuously has to update the matrices without using any of the additional information in the system. Therefore is chosen to end the MCTS when the parked trains are leaving.

The information will be retrieved from the matrices by analyzing which parking spot is available per track, considering that the trains will be parked consecutively. Then, only the first consecutive free parking spot will be used. This means that there is only one parking spot per track considered. This track and parking spot refers to a specific row of the matrices, the columns of the matrices refers to individual trains. From these matrices the Robust max-child can be created by dividing the matrices through each other. The Robust max-child approach uses the deviation between the number of feasible solutions and the total number of passes (Chaslot, 2010). This resulting matrix is used to extract the data for a train, considering the earlier formulated parking spots. This means that a list with length n is subtracted where n

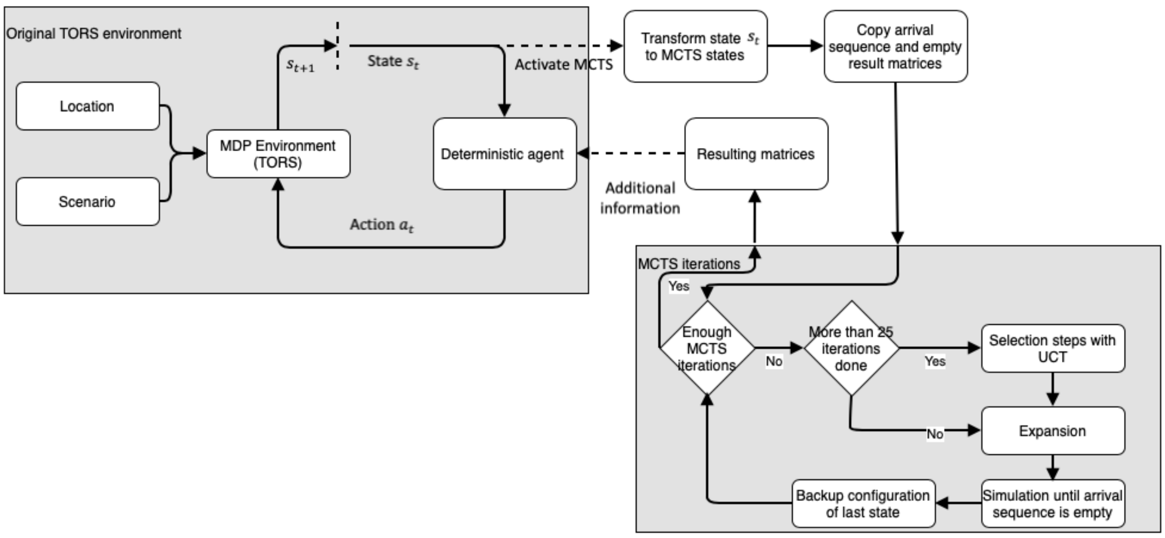


Figure 4.2: Flow chart of the implementation of the Monte Carlo Tree Search.

is equal to the number of considered tracks. This list contains multiple fractions normalized through a min-max normalization to determine the underlying importance of the considered tracks with the specific spots. Then, when the deterministic agent assigns a priority value to a move action, additional information in the subtracted list is added to the priority value. If the MCTS are not able to find any good solution, the normalized values are 0. This means that the MCTS is neglected, and no distinction between tracks will be made. Then still an random choice between the different tracks will be made.

There are some differences when using the MCTS approach for the service tracks or other shunting yard types. The first difference for the service MCTS is the arrival sequence. Since not all trains need service, it does not make sense to use the complete arrival sequence. Therefore, the arrival sequence is changed to the trains that need service which already have entered the shunting yard. When a service action is finished, the service time becomes zero and that train can leave the service tracks and will not be considered in the MCTS anymore. The differences between a carousel shunting yard and a shuffleboard shunting yard can be noticed in how the trains are parked. In the MCTS state space for both locations, the trains are always parked closest to the side that is giving the shortest way to the gateway track. In a carousel-like shunting yard with free tracks, the trains are forwarded to the beginning of the tracks. With LIFO-tracks, on a shuffleboard shunting yard, the trains move when another train arrives at the same track because the side of the track where the trains arrive is the side of departure. This means that when two trains are on a track and a third will be newly parked, both parked trains move to make room for the new train. In this manner, the MCTS methods can be applied to both types of tracks. The implementation of the MCTS is depicted in the flow chart Figure 4.2. Besides the elaborated basic implementation, some speed-up methods are implemented that will be elaborated in the following paragraph.

4.2.2 Increasing computation efficiency

The MCTS follows a tree policy in the selection step. This tree policy normally follows the highest-scoring trajectory that is visible in both the matrices. This means that the best scoring trajectory is exploited. To improve on this mechanism and to improve the speed of convergence, an Upper Confidence Bounds for Trees is introduced by Kocsis and Szepesvári (2006). This method is based on the multi-armed bandit Upper Confidence Bound algorithm developed by Auer et al. (2002) but then formulated to a tree structure. The Upper Confidence Bounds for Trees is calculated on all possible actions, and the highest valued confidence bound is chosen because that node gives the highest potential value and therefore gives the best result if repeated enough. Equation 4.3 gives the formula for calculating the Upper Confidence Bounds for Trees, and Equation 4.4 will be used to select the best node. Here $w_{(s,a)}$ is the number of successful moves of the considered child node. The values $N_{(s)}$ and $n_{(s,a)}$ are the number of simulation passes for the parent and child node respectively. The first part of the formula $(w_{(s,a)}/n_{(s,a)})$ focuses on exploiting the current policy, and the last part of the formula focuses on the exploration of other, better, policies. The exploration and exploitation are balanced naturally through the design of the formula but can be modified by the exploration parameter $c > 0$ if necessary. This parameter is theoretically equal to $\sqrt{2}$, and a higher parameter means more exploration.

$$UCT_{(s,a)} = \frac{w_{(s,a)}}{n_{(s,a)}} + c\sqrt{\frac{\ln N_{(s)}}{n_{(s,a)}}} \tag{4.3}$$

$$a = \underset{a}{arg\,max}\ UCT_{(s,a)} \tag{4.4}$$

The Upper Confidence Bounds for Trees starts working after the backup matrices are filled to a certain amount. This is because it needs some data before it can actually be used. Then the Upper Confidence Bounds for Trees is followed until a moment is reached where the no deliberated action can be made. If the Upper Confidence Bounds for Trees cannot return an action the selection step will be ended and the MCTS continuous to the expansions step. Another speed-up method is introduced by Gelly and Silver (2007). This method gives a Rapid Action Value Evaluation of action a in state s. This means that the states that follow the tree-policy and the states in the roll out phase are stored. This low variance approach could enable a bias; normally, the bias is counteracted with a decaying weight. However, a bias does not matter in our work since we are not looking for an optimal value but rather a feasible solution; therefore, the decaying weight is not used. This Rapid Action Value Evaluation extension is applied by saving the last obtained MCTS state. Then the information of both the tree policy and the roll out will be considered.

Besides the scientific substantiated approaches, two heuristics are introduced. The first heuristic ensures that the roll out phase of the MCTS is biased to create full tracks. This bias is created by taking the matched number of a train that has to be parked and looking at the matched number of the last parked train of all considered tracks. For every considered track, the difference in matched numbers is calculated, and the positive value closest to 0 is the best place to park the train behind. If there is no positive value, then the train is parked on a new track. The second heuristic takes the combination action into account. If a train, which is already parked, needs to be combined with another train, which is not yet parked, it cannot create a feasible solution in the roll out phase unless these trains are parked behind each other. This forces the MCTS to converge to solutions where the combined trains are parked behind each other. When the sequential parking is successful, the MCTS continuous normally.

4.2.3 Additional relocation heuristic

When analyzing the results of the initial measurement, a potentially helpful optimization approach was discovered. It became apparent that for the carousel-like shunting yard, many scenarios fail because of non-deliberated early moves. More specifically, if the first train that arrives at a shunting yard has match number 16, then the most optimal track is being chosen for that train. After that, a second train, with match number 15, arrives the most optimal track is chosen again. Unfortunately, the most optimal track for the second train is a new track because it cannot leave the shunting yard if parked behind the train with a higher match number. Therefore, two tracks are occupied with only one train each, which is suboptimal. This situation worsens when considering that match number 16 is the highest matched number that should arrive at the shunting yard. To overcome this problem, another heuristic is implemented. This heuristic ensures that when there are more than two tracks occupied, with a single train on at least one of the tracks, a relocation action is proposed, such that the MCTS can find a better-suited parking spot. When more than one tracks have a single train on it, the train with the highest matched number will be relocated. To enable the relocation, the implemented algorithm is depicted (algorithm 3). The effect of this extra heuristic only works when using it in combination with the MCTS. This is because, the deterministic agent will assign the same priority values, which will result in a similar parking composition. This means that if the MCTS is not used the TORS environment only moves backward and forward without changing the situation.

Algorithm 3: Improved relocation heuristic,

```
Input: Track layout $k \in \mathcal{K}$,
Output: Increased relocation reward

$u_{reloc} = 0$
for $k = 1, 2 \ldots, \mathcal{K}$ do
    $u_{position}$ = position of most right train in $k$
    if $u_{position} = 1$ then
        if $u_{reloc} > u_{position}$ then
            $u_{reloc} = u_{position}$
end
if $u_{reloc} > 0$ then
    Increase relocation reward for train $u_{reloc}$
```

Chapter 5

Experimental Setup

In this chapter, the details of the experiments are elaborated. The experiments are divided into two parts, one for each part of the framework. First, a description of the data is given. Then the experiments for the hyperparameter optimization will be elaborated and subsequently, the experiments that focus on handling the non-stationary variables will be elaborated. The scenarios and specifications of the locations will be similar for the experiments in both parts of the framework. This chapter ends with a description of the benchmark algorithm used in the experiments.

5.1 Data generation

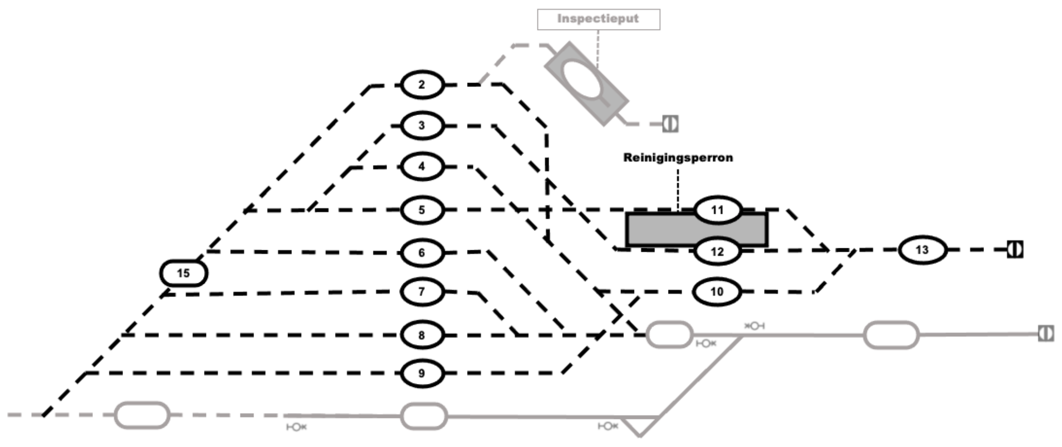
The TORS environment can simulate the creation of a shunting plan on every shunting yard in The Netherlands. To create a shunting schedule, two files are necessary, these are the location and the scenario files. This subsection will elaborate on the locations and scenarios that will be used throughout all the experiments.

5.1.1 Locations

The experiments will be limited to two locations. The first location is Kleine Binckhorst and the second location is a shunting yard named Idealized Location. These two locations are chosen because of the different types of shunting yards that require a different planning approach. The Kleine Binckhorst is a carousel-like shunting yard close to The Hague central station and is one of the smallest shunting yards of the NS. Moreover, this shunting yard gives a realistic aspect on the experiments. Therefore, this shunting yard is often used for research purposes (van den Broek, 2016; Lee et al., 2020). The Idealized Location is a small non-existing shunting yard containing a shuffleboard-like layout. Each location will be elaborated on in detail.

Kleine Binckhorst

The shunting yard Kleine Binckhorst consists of thirteen different tracks (Figure 5.1). This shunting yard contains one gateway track denoted with number 15, eighth parking tracks, indicated with numbers 2 to 8. Two dedicated service tracks are denoted with numbers 11 and 12 and finally, one dedicated relocation track, represented with number 13. Track number 10 is used in the TORS environment, but this track is only used as a transfer track to enable the routing. All the routing possibilities and the dedicated lengths for each track can be seen in Figure 5.1.

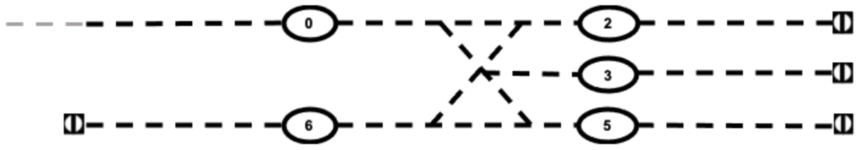


Track	2	3	4	5	6	7	8	9	11	12	13	15
Track length (m)	480	399	387	357	222	202	203	182	247	247	271	275

Figure 5.1: Graphical representation of the shunting yard Kleine Binckhorst, darker parts are only considered by the TORS environment (retrieved and adjusted from www.sporenplan.nl). The lengths of the tracks are given in the table accordingly.

Idealized location

The second location is the Idealized Location, this is a shunting yard that consists of five tracks. There is one gateway track, three parking tracks, and one relocation track denoted with 0, 2, 3, 5, 6, respectively. All these tracks and the dedicated lengths are visible in a graphical representation (Figure 5.2). The difference with the shunting yard, Kleine Binckhorst, is that there is no dedicated service track which means that service scheduling will not be considered in these experiments. This also means that the service action will be neglected in these scenarios.



Track	0	2	3	5	6
Track length (m)	400	500	500	500	500

Figure 5.2: Graphical representation of the idealized shunting yard, the darker parks are considered in the simulation model. The lengths of the tracks are given in the table accordingly.

5.1.2 Instance generator

Each scenario contains the arrival and departure sequences that will be simulated. These sequences cannot be made randomly, this is because the incoming trains need to be aligned with the outgoing trains. Besides that, a train cannot depart at time t while the planned arrival is at $t + 1$. Therefore, an instance generator is used. This instance generator is developed within the NS to make real-life scenarios by balancing the arrival and departures and ensuring reasonable time for service activities. Throughout all the instances that will be used in our experiments, all the trains arrive before the first train's departure, and like mentioned before, the maximum number of train units in a departing train is two. Both are done to control the complexity of the instances. The scenarios for the location Kleine Binckhorst are different from the idealized location. Therefore, the scenarios for each location will be described individually. The trains considered in both locations are the SNG, SLT, and VIRM trains, with three and four carriages for the SNG trains and four and six carriages for the SLT and VIRM trains. For all these trains the lengths are given (Table 5.1).

Table 5.1: Length per train type and subtype

Train type	SNG-3	SNG-4	SLT-4	SLT-6	VIRM-4	VIRM-6
Length (m)	60	76	70	101	109	162

The scenarios of the different locations cannot be compared to each other. This is because these scenarios are specifically generated per location and each location has its own specifications, with the associated complexity. For example, the shunting yard Kleine Binckhorst contains more tracks and all tracks together are longer if compared with the Idealized Location. Besides that, the location Kleine Binkhorst consists of service tracks and the Idealized location not. The number of train units per location is determined by balancing the complexity between the scenarios on a similar location. With this, all scenarios must be reasonably complex to be solved by the deterministic agent. This means that all trains fit on the shunting yard, but no experiment will have a feasibility performance that is near 100%, without any applied optimization techniques. Therefore, each location consists of three types of scenarios, all with different quantities of train units. The experiments for the location Kleine Binckhorst contain 15, 16 and 17 train units and the experiments for the Idealized Location contain 10, 12 and 14 train units.

Scenarios for the Kleine Binckhorst

The scenarios used in the experiments for the Kleine Binckhorst consist of 15, 16, and 17 train units named scenarios A, B, and C, respectively. With scenario A, the total train units are equal to the outgoing trains, indicating no combination is required, only splitting. This is not the case for scenarios B and C, here trains have to be split and combined. If the incoming train units is equal to the outgoing train units, still a split and combine action is required. Throughout these scenarios, the train unit types are set to SLT and VIRM with subtypes of four and six. The distribution of types is given for all number of trains units considered in the scenario (Table 5.2). Train units of a similar type can be combined in a departing train. The sets of departing train units for the SLT train types are {SLT-4, SLT-4} and {SLT-4, SLT-6}. Equally, for the VIRM train unit types, the sets {VIRM-4, VIRM-4} and {VIRM-4, VIRM-6} can be made. Besides these combinations, every train type can be a departing train individually. The scenarios of the shunting yard Kleine Binckhorst include a representation for the service times. A representation of a complete scenario of scenario type C is visible in Table 5.3.

Table 5.2: Composition of three scenarios for the location Kleine Binckhorst with the total train units, the incoming train units, the outgoing train units and the number of train units per type.

	Number of train units			**Number of train units per type**			
Scenario	**Total**	**Incoming**	**Outgoing**	**SLT-4**	**SLT-6**	**VIRM-4**	**VIRM-6**
A	15	13	15	3	3	5	4
B	16	14	15	3	3	6	4
C	17	16	16	4	4	5	4

Table 5.3: A shunting scenario, generated by the instance generator. A scenario contains events that enable the arrivals and departures of trains at a shunting yard. These arrivals and departures correspond to one or more train units that subsequently correspond to a service time and match number.

Event Number	**Event type**	**Time of event**	**ID(s)**	**Display name(s)**	**Service time(s)**	**Matched number**
0	In	51074	2601	SLT6	0	1
1	In	40357	9401	VIRM4	1380	12
2	In	48744	2602, 2401	SLT6, SLT4	780, 540	9, 6
3	In	47214	9402	VIRM4	1380	15
4	In	50713	2603	SLT6	0	14
5	In	57037	2604	SLT6	0	8
6	In	40820	9403	VIRM4	1380	5
7	In	58876	2402	SLT4	540	2
8	In	39997	9404	VIRM4	1380	4
9	In	59625	8601	VIRM6	2100	13
10	In	59264	8602	VIRM6	0	3
11	In	61447	9405	VIRM4	1380	11
12	In	63576	2403	SLT4	540	16
13	In	57945	2404	SLT4	540	1
14	In	49104	8603	VIRM6	2100	7
15	In	45777	8604	VIRM6	0	10
16	Out	77632	2601, 2404	SLT6, SLT4	NA	1
17	Out	77271	2402	SLT4	NA	2
18	Out	78768	8602	VIRM6	NA	3
19	Out	79489	9404	VIRM4	NA	4
20	Out	84908	9403	VIRM4	NA	5
21	Out	80569	2401	SLT4	NA	6
22	Out	77992	8603	VIRM6	NA	7
23	Out	82654	2604	SLT6	NA	8
24	Out	84477	2602	SLT6	NA	9
25	Out	83716	8604	VIRM6	NA	10
26	Out	79128	9405	VIRM4	NA	11
27	Out	79849	9401	VIRM4	NA	12
28	Out	82098	8601	VIRM6	NA	13
29	Out	80209	2603	SLT6	NA	14
30	Out	81012	9402	VIRM4	NA	15
31	Out	85337	2403	SLT4	NA	16

Scenarios for the Idealized location

The experiments for the idealized location consist of scenarios with 10, 12, and 14 train units referred to as scenarios D, E, and F, respectively. With these scenarios, the number of incoming trains and outgoing trains differ per scenario. This is visible in Table 5.4. Therefore contain all the scenarios split and combine actions. In these scenarios, the train unit type SNG has a subtype of three and four and the SLT and VIRM types have a subtype of four and six. With each number of train units, a unique distribution of the train type and subtypes are given (Table 5.4). In the scenarios of the idealized location, every number of trains will contain train units that have to be combined to match a departing train. The combinations that can be made in the idealized location are in the sets of {SNG-3, SNG-3}, {SNG-3, SNG-4}, {SLT-4, SLT-4}, {SLT-4, SLT-6}, {VIRM-4, VIRM-4} and {VIRM-4, VIRM-6} for the SNG, SLT and VIRM types respectively. In contradiction to the scenarios generated for the shunting yard Kleine Binckhorst, the idealized location scenario does not consider service times.

Table 5.4: Composition of three scenarios for the location Idealized Location with the total train units, the incoming train units, the outgoing train units and the number of train units per type.

	Number of train units			**Number of train units per type**					
Scenario	**Total**	**Incoming**	**Outgoing**	**SNG-3**	**SNG-4**	**SLT-4**	**SLT-6**	**VIRM-4**	**VIRM-6**
D	10	8	6	2	2	2	1	1	2
E	12	9	6	2	2	2	2	2	2
F	14	11	8	2	2	3	2	2	3

5.2 Experiment description for hyperparameter optimization

This section will describe the experiments related to the first part of our framework, namely the hyperparameter optimization. This is done by elaborating the design of the experiments, the parameter setting and performance metrics.

5.2.1 Experimental design

The experiments related to hyperparameter optimization will be executed on the deterministic agent without any applied enhancements. The experiments described in this section are executed for both locations. Throughout these experiments, contextual information is considered which is the deviation of the scenarios that are formulated in the previous section.

Before the Bayesian Optimization framework can be used, some information must be present to compute the surrogate model. Therefore, each hyperparameter optimization method will start with random iterations not selected through the Bayesian Optimization technique. Each hyperparameter optimization approach will begin with nine random iterations. After these nine iterations, it is still possible that no feasible solutions are obtained. Therefore, we will investigate if a warm start method will provide an advantage to our optimization model. Lindauer and Hutter (2018) described an approach for warm starting a model-based algorithm configuration. This approach is named warm starting Initial Design (INIT) and will consider selecting a predefined hyperparameter set that has proven to give reasonable results. The advantage of such a start procedure is that it substantially speeds up the hyperparameter optimization but can introduce a bias to the search space. The hyperparameters that have proven to give reasonable results are the priority values it is originally designed with. Therefore, these hyperparameters will be used throughout this experiment. Besides, the effects of a bias in the search space are not considered a problem since it is reasoned that the optimal solution contains similarities with the default hyperparameter set.

After the random iterations are executed, the number of Bayesian Optimization iterations has to be determined. This number determines how often the optimization cycle is repeated. Selecting the right number of iterations is complex since too many iterations create inefficient computation times while the quality of the obtained solution can decrease. The quality of a solution can decline with a larger number of iterations since the model searches for a hyperparameter configuration that is optimal on the training data, which cannot be generalized to the test data. Formulated otherwise, the Bayesian Optimization is over-fitting on the training data. Contrarily, early stopping the iterations might be possible in some situations and can improve the quality of a found hyperparameter solution. However, then it might be hard to determine the reliability of a solution (Falkner et al., 2018). Since there is no strict manner to determine the right number of iterations, this is determined empirically. The empirical evidence will be gathered through experiments for every location, with these experiments the Bayesian Optimization is executed with a large number of iterations. Every time the Bayesian Optimization finds an improved hyperparameter configuration, it is tested on the test set. By keeping track of the training and test results with a large number of iterations the convergence rate of the Bayesian Optimization approach can be noticed. Eventually, after a large number of iterations the Bayesian Optimization procedure is not able to improve the training and test results further. When that moment is reached, the required number of iterations for convergence empirically be determined. This will be done by selecting the number of iterations in which the Bayesian Optimization has enough time to converge without creating inefficient computation times. The maximum number of iterations for the Bayesian Optimization is set to 250 iterations and these experiments are executed with and without a warm start procedure. This warm start procedure is added to see if convergence can be achieved faster. For these experiments, the training and test batch sizes are set to 80 scenarios.

For the hyperparameter optimization, a division must be made between a training and test set. From this training set, a batch is taken that is used to train the model. When the optimal hyperparameter configuration is found, a similar-sized batch is taken to test the model, this will be referred to as the test set. Distributing the data in a training and test set is done with a holdout method. This means that the scenarios in the test set are extracted from the training data, these cannot be seen during the training phase. Both the training and test data require a batch of scenarios. Determining the size of these batches is a delicate procedure, often in machine learning problems, a too small batch size is chosen. This leads to a poor generalization to the test data and introduces the possibility to overfit on the training data. However, selecting a too large batch size can create inefficient experiments due to the exhaustive computation times. Therefore, a smaller batch size is cheaper in terms of evaluation but will result in less accurate predictions.

There are no standardized approaches to determine the right batch size, therefore will the batch size be determined empirically. To empirically determine the right batch size, an experiment is created that executes a set of Bayesian Optimization iterations with an increasing batch size per experiment. For both locations, batches of 64, 80 and 96 scenarios are created. These numbers are chosen because the used machines can simulate eight TORS environments simultaneously. Besides that, it is expected that the large deviation between lowest and the highest batch size gives a clear representation of over-fitting. To increase the robustness and prevent over-fitting even more, a new training batch is sampled with every Bayesian Optimization iteration, which is a common approach (Hutter et al., 2013).

After determining the right number of iterations and the right batch sizes, the experiments of the Bayesian Optimization are executed. As formulated earlier, the contextual information represents the number of train units considered in a scenario. As can be seen in tables Table 5.2 and Table 5.4 the scenarios give insight into the difference in complexity. The quantities of the incoming and outgoing trains differ with the total train units, this indicates split and combine actions that require a more complex planning strategy. Besides that, the train types considered in each scenario differ, indicating different lengths of trains and different strategies to create a shunting plan.

The following two experiments focus on analyzing the importance of contextual information with Bayesian Optimization. The first experiments use the Bayesian Optimization and compare it with a TORS environment with the priority values it is originally designed with. This original priority values will be referred to as the *default hyperparameters* and can be found in Table 5.5. The comparison with these default hyperparameters is made because these parameters represent the best scenario that could be created through the intuition of human planners. Both the training and test data do not consider any contextual information for both experiments, which means that no distinction is made between different scenarios. The second experiments considers the contextual information per scenario. Here, multiple smaller experiments are created based on the differences that can be found in each scenario. Per scenario, an experiment will be used to show the difference between the default hyperparameters and the Bayesian Optimization approach to determine the hyperparameters.

Table 5.5: Priority values of the initial designed deterministic agent per action and per location.

Experiments	**Connect**	**Depart**	**Move entering**	**Move internal**	**Service**	**Setback**	**Split**	**Wait**
Kleine Binckhorst	280	400	200	100	360	200	380	100
Idealized Location	320	400	300	100	NA	260	320	100

5.2.2 Defining the search space

Eight different hyperparameters need to be optimized. Each hyperparameter equals the priority value of an action in the TORS environment and the movement action will be divided into two hyperparameters. All these hyperparameters together form the search space. Throughout this research, we want to give the Bayesian Optimization model the opportunity to search all possible priority configurations. Therefore, all these hyperparameters are tested in equal finite domains, with an under bound of 0 and an upper bound of 400. The upper bound is set to 400 because that is the maximum value specified in the default hyperparameters and therefore can similar ranges be guaranteed.

5.2.3 Objective and performance metrics

The objective and performance metrics are elaborated for the hyperparameter optimization. These performance metrics can be used to compare the results of the experiments between the different experiments and obtain the best hyperparameter setting.

First-time-right performance - The first-time-right performance relates to the objective formulated in chapter 3. To repeat on that shortly, the first-time-right performance is when a scenario has been solved without any violation of the constraints. A scenario has been solved when all the arrivals and departure events formulated in the scenario file can be executed.

After the first-time-right performance is calculated, the difference between the optimized hyperparameters and the default hyperparameters will be statistically determined through a t-test (Student, 1908). With this test is determined if the difference between the two means is sufficient to be called significant.

5.3 Experiment description for handling stochastic variables

This second part will give a detailed description of the experiments regarding the non-stationary variables. First, the design of the experiments will be elaborated. After that, the parameter settings and the performance metrics are formulated.

5.3.1 Experimental design

For the experiments regarding the handling of the non-stationary variables, experiments are created. The first set of experiments consist of the default hyperparameter configuration. The default hyperparameter configuration consist of the priority values that the NS used while designing the TORS environment. These experiments are applied to the locations Kleine Binckhorst and the Idealized Location, each with their own scenarios that will be tested. For the location Kleine Binckhorst three different experiments will be created;

- *Plain heuristics* - With this experiment, the TORS environment with the deterministic agent is used without any improvements. This experiment is created to set a performance baseline.
- *With MCTS* - Within this experiment, the TORS environment with the deterministic agent is extended with the MCTS.
- *With MCTS and relocation* - With this experiment, the TORS environment with the deterministic agent is extended with the MCTS in combination with the advanced relocation heuristic.

For the Idealized Location are the first two experiments used. The third experiment is not relevant since the deterministic rules for the Idealized Location are designed to take more complex relocation movements into account. The Idealized Location requires more complex relocation actions because the location is designed with fewer but longer tracks. Besides that, the Idealized Location only consist of LIFO-tracks.

Eventually, the model that reaches the highest feasibility rate on all the scenarios will be compared with a benchmark algorithm. This benchmark algorithm will be elaborated in section 5.4. This benchmark algorithm will be tested on the formulated scenarios for both locations.

For these experiments, the data does not have to be divided into a training and test set. In this case, it is more important to guarantee the accuracy of the performance metrics. This is done by calculating the number of simulation runs suited to create a certain margin of error. To calculate the number of runs per experiment, the Central Limit Theorem is used (Boon et al., 2020). Central Limit Theorem provides a means for determining the number of iterations for a simulation model to give an accurate performance metric. This is done by estimating σ by doing a small number of simulation runs. In our research, to calculate σ, 25 simulation iterations are done. With this σ the required number of iterations can be calculated to obtain a $(1-\alpha)\%$ confidence interval based on the margin of error ϵ. The formula to calculate the number of runs is given in Equation 5.1. For our research, we want to obtain experiments with a confidence of 95%, with a margin of error of 3%.

$$n > \left(\frac{Z_{\alpha/2} \cdot \sigma}{\epsilon}\right)^2 \tag{5.1}$$

Due to many experiments, only a single group of experiments is used to calculate the confidence and accuracy levels. This will be the scenario with 17 train units for Kleine Binckhorst's location because this is the most complex scenario. For the experiments, *plain heuristics, with MCTS* and *with MCTS and relocation* the number of iterations are 1,041, 1,060, 1,010 respectively. To compensate a little for the exhaustive computation time, the number of iterations n is set to 1,000. This means that the accuracy increases from 3% to around 3.1%. With an accuracy of 3.1% the number of iterations becomes 988, 997 and 951 for the experiments, *plain heuristics, with MCTS* and *with MCTS and relocation*, respectively.

5.3.2 Parameter settings

The MCTS has multiple parameters that can be used to guide the search process. First, the computational resources. The MCTS uses computational resources to determine the number of MCTS iterations. Commonly, the computational resources are expressed by a computation time in which the MCTS must be able to generate as many results as possible. When such an approach is used, the computation time needs to be long enough to gather enough information but does not need too long such that unnecessary iterations are done. To overcome this dilemma, the number of iterations will be used instead. Therefore the computational resources are replaced by a counter that counts the number of iterations and successful iterations. When the MCTS for the location Kleine Binckhorst reaches 1,000 successful iterations or 10,000 non-successful iterations, the procedure terminates, and the filled table can be used to obtain the best possible move. For the Idealized Location, this is set to 750 successful iterations and 20,000 non-successful iterations. The difference between the parameters per location originates from complexity in the scenarios that arise through the split and combination actions. The number of splits and combine actions is higher on the Idealized Location since all scenarios require more split and combine actions and fewer tracks are available to execute these actions.

The MCTS has a hyperparameter to balance the exploration and exploitation trade-off, for this research, this parameter is set to 0.1 to focus more on the exploration. This parameter is set to a low value because the RAVE speedup technique ensures that the values are exploited enough.

5.3.3 Objective and performance metrics

The objective and performance metrics for these experiments will be used to compare the different results and eventually compare them to benchmark studies. The performance metrics are;

1. *First-time-right performance* - The first-time-right performance relates to the objective formulated in chapter 3. To repeat on that shortly, the first-time-right performance is when a scenario has been solved without any violation of the constraints. A scenario has been solved when all the arrivals and departure events formulated in the scenario file can be executed.
2. *Violations* - This performance metric keeps track of the violations that can occur in the TORS environment. Every time a shunting plan is infeasible is counted which violation has been violated. These violations can give insights into the quality of the used extension of the deterministic agent. The violations considered throughout this work are formulated in section 3.1.3.
3. *Computation times* - The computation time for all experiments are tracked. These can be used to compare the different methods. The computation time differs per location, this is dependent on the number of available actions, which is higher in the shunting yard Kleine Binckhorst. However, this also depends on the computational power of the used machines. Throughout this thesis, two different machines are used, and since the shunting yard Kleine Binckhorst is more expensive to compute, a heavier machine is used. The heavy machine contains an Intel Core i7-4710HQ 4th Gen quad-core 2.5-3.5GHz 6MB Cache with 16 GB of RAM and the lighter machine, which is specifically used for the Idealized Location, contains an Intel Xeon Platinum 8272CL-processor dual-core 2.5-3.4GHz with 16 GB of RAM.

5.4 Benchmark algorithm

A benchmark algorithm is used to give a comprehensive view of the experiments and compare the solutions. The benchmark used throughout this research is the local search algorithm developed by van den Broek (2016), named HIP. This benchmark algorithm uses a local search model to improve some candidate solutions gradually. The candidate solution is formulated as an activity graph that represents the shunting schedule. This graph is then used in the simulated annealing, a stochastic variant of a local search algorithm, to make local changes to the schedule and create a feasible shunting schedule. A graphical representation of that approach is given earlier (Figure 2.2). This approach is currently implemented in the software tools to support the planners of the NS. This benchmark algorithm can be used with the earlier defined location and scenario files. Therefore an equal difficulty in scenarios can be guaranteed. However, it should be taken into account that the way of generating the shunting schedule differs. The local search algorithm generates a shunting plan by analyzing the solution holistically. If a disturbance occurs, the complete solution has to be rescheduled and no information about the old solution is maintained. This makes the local search approach less suitable for realistic scenarios that experience disturbances because it will be inefficient to reschedule the complete solution with disturbances continuously. That is the main difference with our approach, the considered deterministic agent can create a shunting plan while executing, which prevents the rescheduling of the solution with disturbances. Consequently, the violations or computation times cannot be compared directly, but only, the number of feasible solutions will be compared to indicate the performance against the benchmark.

Chapter 6

Results

This chapter will elaborate on the results obtained when implementing the optimization framework to increase the feasibility rate of the TORS environment with the deterministic agent. The first part shows the results of the hyperparameter optimization, which will be used to configure the optimal deterministic agent. After that, the results of the second part of the optimization framework are elaborated, focusing on enhancing the deterministic agent's search capabilities.

6.1 Hyperparameter optimization

Before the results of the hyperparameter optimization are formulated, the robustness of the analyses is determined. This robustness is determined by selecting the right training parameters. The training parameters consist of choosing the optimal number of iterations necessary for convergence and selecting the right batch size to prevent overfitting the results. After that, the results of experiments regarding the context-free and contextual hyperparameter optimization methods are elaborated, and finally, the best hyperparameter configuration is selected which is used throughout the resulting experiments.

6.1.1 Training parameters

There is no standardized approach to determine the number of Bayesian Optimization iterations necessary for a robust experiment. Therefore, the first set of experiments will be used to empirically determine the stopping criteria of the Bayesian Optimization repetitions. The stopping condition determines at which point to stop the search for better fitting solutions. Choosing the correct value is a delicate procedure because a value too small will result in a flawed model that will not converge to a better solution, and a value too large can result in overfitting the model. Besides that, this section elaborates the results when the warm starting Initial Design (INIT) approach is added to the model. The INIT approach uses a predefined hyperparameter set that has proven to give the reasonable result to start a model-based algorithm configuration approach. The predefined hyperparameters that have proven to give reasonable results are the hyperparameter that are used while designing the deterministic agent. Therefore, these hyperparameters will be used in the warm starting procedure.

This experiment is executed as follows, every time the BO finds an improved number of feasible scenarios, the hyperparameters configuration is stored and these hyperparameters are tested against the test set. Then, the feasibility rates, of the training and test data, are stored and the BO procedure continues until another improvement in the number of feasible scenarios is found. This iterative procedure is repeated until 250 different hyperparameter configurations are analyzed. After that, the rate of convergence will be analyzed and the number of iterations will empirically be determined. For these experiments, the batch sizes of the training and test data are set to 80 scenarios. These experiments are executed for the location Kleine Binckhorst and the Idealized Location with the experimental setups B and E, respectively. These are used because these experiments have a medium complexity compared with the other experiments on similar locations. Therefore, this will generate the most divisive and robust results. Logically, the deterministic agent is used without any additional enhancements.

The results of the experiments are given for both locations (Figure 6.1). The solid lines in both graphs indicate the BO iterations and every time the model finds a better solution, the line increases to that value. The dotted lines represent the test results. The color differences show the difference between the warm start approach and the regular Bayesian Optimization approach.

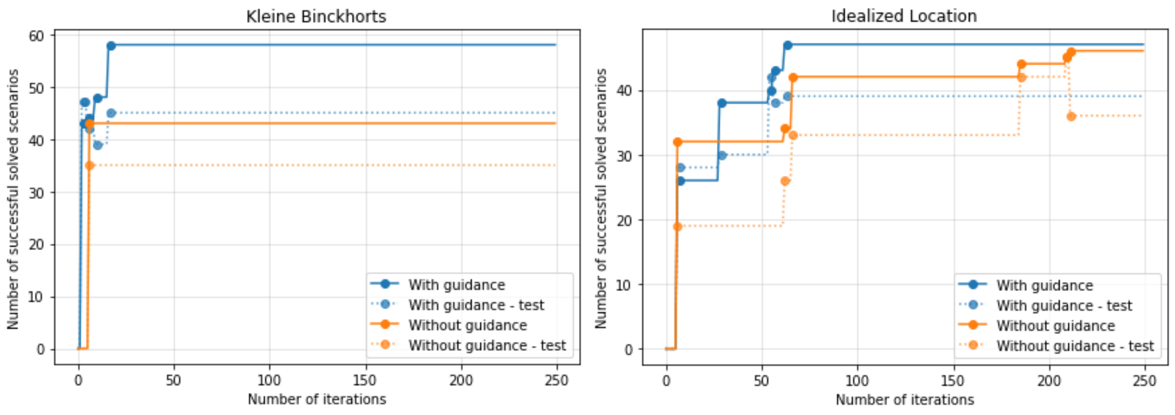


Figure 6.1: Number of iterations to convergence, with and without guidance for training and test instances. Guidance means the first Bayesian Optimization iteration with the default hyperparameter configuration.

The results show that the training phases of both locations can create improvements when the number of iterations increases. The location Kleine Binckhorst gives only a single improvement when considering the Bayesian Optimization without the warm start procedure. A rationale for this is that only eleven solutions were found that did not returned zero feasible scenarios in the whole experiment without the warm start approach. When using the warm start procedure, more than 100 solutions were found that did not return zero. The first eleven hyperparameters for both locations are depicted Appendix A. After an in-depth analysis it became apparent that the hyperparameters of the Kleine Binckhorst have to be better aligned with each other to create a feasible solution. This means that the hyperparameters of the Idealized Location have to be less accurately aligned. A reason for this can be found in the design of the shunting yard. The Idealized Location requires more relocation movements and therefore the timing of certain actions (e.g. the timing of a connect action) has to be less accurate than on the Kleine Binckhorst since there are multiple moments to execute the action. Formulated otherwise, for the Idealized Location the deterministic agent can simply keep relocating trains until an action becomes feasible and then the action can be executed. Therefore, the feasible region, in the configuration space of all the hyperparameters, is smaller and thereby harder to be found for the Kleine Binckhorst when not using any guidance.

When using the warm start procedure on the Kleine Binckhorst, multiple improving hyperparameter configurations are found that show improving results on the test data. This means that the found hyperparameter set can be generalized to the test set. Therefore can be concluded that the warm start approach has the desired effect for the location Kleine Binckhorst. The results are improved when using the warm start approach while the computation times remain efficient. Since the hyperparameters do not improve after 19 Bayesian Optimization iterations is decided to set the maximum number of iterations for the experiments related to the location Kleine Binckhorst to 30 iterations.

Then the results for the Idealized Location, the BO approach for the idealized location, is depicted in the right figure (Figure 6.1). The improvements without the warm start approach show a gradual increase in training results. This improvement can also be found in the test results. When the warm starting procedure is added, similar gradual improvements can be found, but the time frame in which the results are gathered is substantiated shorter, this while the found solutions can be generalized to the test data. Therefore can be concluded that the warm start approach has the desired effect for the Idealized Location. This warm start extension will therefore be used throughout the following experiments. The last improvement iteration for the Idealized Location with a warm start procedure is found after 68 iterations. Therefore is decided to set the number of iterations for the experiments related to the Idealized Location equal to 80.

The second set of experiments determines the number of scenarios within a batch used for the training and test data. This batch size is determined empirically since this is a complex procedure because too few scenarios in a batch mean that the solution cannot be generalized on the test data. On the other hand, a batch size that is too large will not overfit the data but is suboptimal since it adds expensive computation time. To test the generalization, an experiment with an increasing number of batch sizes is trained and tested. The experimental setup for the Kleine Binckhorst location consists of the BO with batch sizes of {48, 64, 80, 96} and for the Idealized Location, these batch sizes become {64, 80, 96}. By including the 48 scenarios in a batch, for the location Kleine Binckhorst, the model is empirically tested on a batch size with more efficient computation times. This difference is made because the computation times of the Kleine Binckhorst are substantiated larger than for the Idealized Location. The training and test results will be averaged over eight different runs to ensure stability in the results. This experiment will be executed with the deterministic agent without enhancements for the location Kleine Binckhorst and the Idealized Location with experimental setup C and E, respectively. These setups give a good indication of the right batch size considering the complexity of all the instances. The results of the training and test data are presented (Figure 6.2).

Both graphs give the percentage of solved instances per batch size. The left chart displays Kleine Binckhorst, and the right graph shows the results for the Idealized Location. The points are averaged over eight runs where the error regions show the size of the standard deviation. Note that the number of Bayesian Optimization iterations is 30 and 80 for the Kleine Binckhorst and Idealized Location, respectively. With both experiments, the warm starting approach is used. By using the warm starting approach, an under bound is created that creates a certain rate of convergence which minimizes the fluctuations and severity of the error regions.

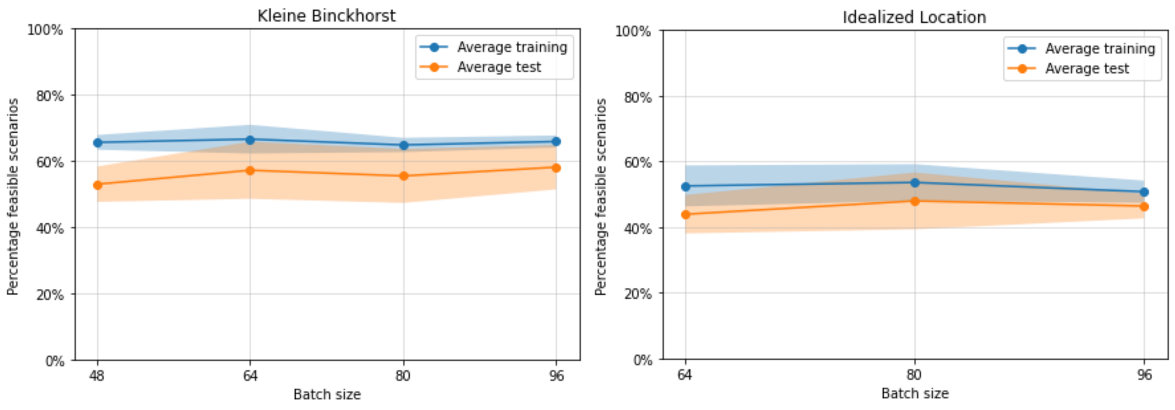


Figure 6.2: Average and one standard deviation of eight training and test data iterations to determine the batch size.

Both locations show that a batch size with fewer scenarios generalizes the solution found on the training data to the test data worse. This means that the smaller batch sizes are more overfitting the training data compared with larger batch sizes. With every batch size for the location Kleine Binckhorst, there is a difference in performance between the training and test results. This is minimized with a training batch size of 64 or 80 scenarios. However, the deviation within the training set with a batch size of 80 is more minor and therefore more stable. Thus, the batch size for the location Kleine Binckhorst is set to 80 scenarios. With the Idealized Location, the training results can be generalized to the test data with the batch sizes that contain 80 and 96 scenarios. The ideal batch size consists of 96 scenarios since that point contains the slightest difference and variation between the training and test data. Therefore, the batch sizes used throughout the following experiments consist of 80 and 96 scenarios for the locations Kleine Binckhorst and the Idealized location, respectively.

6.1.2 Experiments without contextual information

The experiments related to the contextual-fee hyperparameter optimization techniques will be elaborated on first. These contextual-free methods do not consider any contextual information. With these experiments, we tend to show the difference between the hyperparameters used while designing the deterministic agent and the optimized hyperparameters. Besides that, the results of these experiments show if a general hyperparameter set is better to solve the scenarios of the TUSP than specialized hyperparameter configurations. This section tests the hypothesis; an optimized hyperparameter set does not increase performance compared with the default hyperparameters. If the hypothesis is rejected, the BO performs better on both the training and test data, which means that an optimized and generalized hyperparameter configuration leads to more feasible scenarios than the default hyperparameter configuration.

To test this hypothesis, an experiment is created that does not consider any of the contextual information. This means that there is no information available that separates the scenario files. This experiment consists of both the location Kleine Binckhorst and the Idealized Location. To create an experiment that is reliable, each experiment will be the results of 15 different runs. For both the training and test data, statistical evidence is gathered by executing a t-test (Student, 1908). This t-test determines if the difference between the optim-

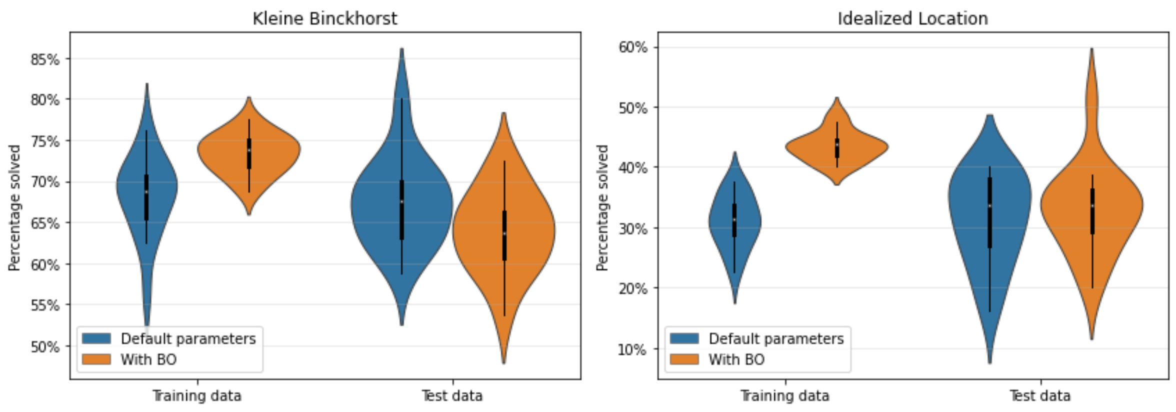


Figure 6.3: Percentage of solved instances with the original hyperparameter setting and with best hyperparameter configuration found by the Bayesian Optimization.

ized hyperparameters and the default hyperparameters is significant, considering the mean of both experiments. Note that a deterministic agent without enhancements is combined with the earlier defined iterations and batch size. The best hyperparameter configuration and the resulting test and training scores are depicted in Appendix C and a summary of the results is visualized in Figure 6.3. The training and test data for the optimized and default hyperparameter sets are displayed through violin plots. Each violin plot gives the distribution of the eight different points. With these plots, a larger size of the violin means a denser region. The center of each violin consists of a miniature boxplot.

The violin plot and the boxplot for the location Kleine Binckhorst distinguish between the default hyperparameter configurations and the optimized hyperparameter configuration. Here, the optimized hyperparameters perform significantly better on the training data ($t(20) = 3.68, p = 0.001$). On the test data the optimized hyperparameters perform significantly worse than the default hyperparameters ($t(28) = -1.85, p = 0.038$). The other violin plot, for the idealized location, clearly distinguishes between the default hyperparameters and the optimized hyperparameters in the training data. This difference is significant ($t(22) = 9.76, p < 0.001$). This phenomenon cannot be observed within the test data. Statistically, the difference in the test data is not significant ($t(28) = 0.62, p = 0.617$). This means that the solution found with BO cannot be transferred to the test set for both locations.

The training results of the Bayesian Optimization are higher than the training results obtained with the deterministic agent and the default hyperparameters. However, the Bayesian Optimization cannot use these improved hyperparameters to create more feasible scenarios on the test data. This means that the model is overfitting the training data. Overfitting on the training data is not a strange phenomenon when there is no contextual information available (Char et al., 2019).

The overfitting in our case is a result of the warm start approach that is being used in combination with the unavailability of the contextual information. This is because the default hyperparameter set is used to warm start the Bayesian Optimization, which creates a lower bound of the results obtained in the training phase. The increase from that lower bound to the actual training results is a result of the different sampled training batches with every iteration. The algorithm currently samples a new training batch with every iteration. This technique is often applied with hyperparameter optimization to reduce the overfitting of a model (Hutter et al., 2013). However, because the contextual information is missing, there is no deviation in complexity, and consequently, this technique can create favorable batches. Char et al. (2019) formulates that if the surrogate model is not able to detect variations in difficulties between certain tasks, then an active task selector does more harm than good. This approach creates a situation where a hyperparameter set that is almost equal to the default hyperparameters in terms of general priority, with a favorable sampled batch makes a substantiated difference in training results. But, these hyperparameters are not able to create improved results on the test instances. This problem could be solved by applying any form of selection while creating the batches. One could, for example, deviate the batches that contain a certain level of complexity. But then, still, some contextual information is considered. In the approaches where contextual information will be used, we do not expect such an occurrence since the difficulty of the sampled scenarios is much more homogeneous.

6.1.3 Experiments with contextual information

The context in the context-based experiments refers to the deviation in the scenarios that are formulated in chapter 5. These experiments aim to show the difference in performance between the optimized hyperparameters and the hyperparameters the deterministic agent is designed with. Through these results can be determined if it would be better to have a unique hyperparameter configuration per context or a generalized hyperparameter configuration that can be used for all considered contextual information. This section tries to answer that through the following hypothesis; an optimized hyperparameter set does not increase performance compared with the default hyperparameter setting. If this hypothesis is rejected, BO creates better solutions on the test data compared with the default hyperparameter setting.

This hypothesis is tested for both the Kleine Binckhorst and the Idealized Location. For both locations, all the scenarios are analyzed with BO and with the deterministic agent fitted with the default hyperparameter setting. As with the contextual-free approaches, each experiment will be repeated 15 times to ensure consistent results, and statistical evidence is gathered through a t-test. The number of iterations, batch sizes, and algorithm type is similar to the previous experiment to create comparable experiments. The best hyperparameter configuration per iteration and the resulting test and training scores are depicted in Appendix C. A summary of these experiments are displayed (Figure 6.4 and Figure 6.5). The results are displayed through different bar charts with whiskers. The bars give the percentage of feasible instances averaged over 15 runs, and the whiskers show the standard deviation for these runs.

We mentioned in the previous section that the experiments that consider contextual information should not overfit the training data, as became apparent with the context-free approaches. When considering contextual information, the complexity within the analyzed scenarios is much more homogeneous. The results for both locations, represented through Figure 6.4 and Figure 6.5, show no overfitting on the training data. The training and test results are not deviating much within these experiments. Therefore can be confirmed that the overfitting of the context-free approaches is mainly due to the lacking homogeneity in the complexity of the scenarios.

The results for the location Kleine Binckhorst are visible in Figure 6.4. In these graphs there is a visible improvement on the training data when using BO. These improvements are significant with, $t(21) = 5.23, p < 0.001$; $t(27) = 2.75, p = 0.005$ and $t(22) = 1.78, p = 0.044$ for the experiments A, B and C, respectively. Interestingly, the standard deviation becomes

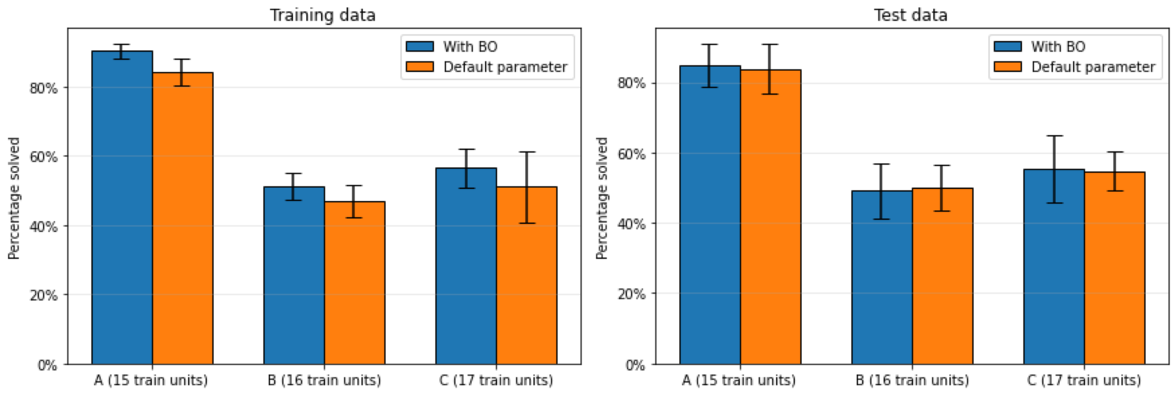


Figure 6.4: Percentage of solved instances for the location Kleine Binckhorst with the original hyperparameter setting and with the best hyperparameter configuration found by the Bayesian Optimization per context, mean and standard deviations are based on 15 iterations.

smaller, which means that the results become more consistent when using BO. This is a result of the sampling strategy that selects new scenarios with every training iteration. Through this method, much more scenarios are considered throughout the complete training phase and therefore results contain less variability. The significant improvement found in the training results is not visible on the test data. Here the best-tested hyperparameter configuration scores similarly to the default hyperparameters, there is a slightly positive difference with experiments A and C, but this is not significant. The student t-test shows that all the differences are not significant on the test data with $t(27) = 0.46, p = 0.326$; $t(27) = -0.32, p = 0.376$ and $t(22) = 0.26, p = 0.397$ for the experiments A, B and C, respectively.

If the best performing hyperparameters are compared with the default hyperparameters can be seen that experiment A obtains in seven iterations similar hyperparameters. This hyperparameter set contains a similar priority as the default hyperparameters. The only difference is that the connect action becomes irrelevant since no combination is required in this experiment. Besides that, experiments B and C show two best performing hyperparameter configurations. These are the default hyperparameters and a hyperparameter configuration that increased the priority of the setback and move entering action and decreased the priority of the split, service and connect actions. This means that the trains are earlier parked than the service is applied which indicates that the carousel movement on the shunting yard is switched to the other direction. This means that the trains are firstly parked and the trains leave the shunting yard through the relocation track. As a result, the setback action becomes one of the most important actions on the shunting yard since a train must be relocated before it leaves the shunting yard.

The results for the Idealized Location are visible in Figure 6.5. When analyzing the results obtained after training the model. The Bayesian Optimization approach can obtain a higher percentage of solved instances per experiment group compared with the deterministic agent with the default hyperparameters. These differences are statistically significant with $t(23) = 2.06, p = 0.026$; $t(20) = 7.41, p < 0.001$ and $t(19) = 3.43, p < 0.001$ for the experiments D, E and F, respectively. Within the training results, the standard deviation becomes smaller when using the BO. Similarly, as for the location Kleine Binckhorst, this results from the sampling strategy that selects new scenarios with every training iteration. Through this method, much more scenarios are considered throughout the complete training phase, and

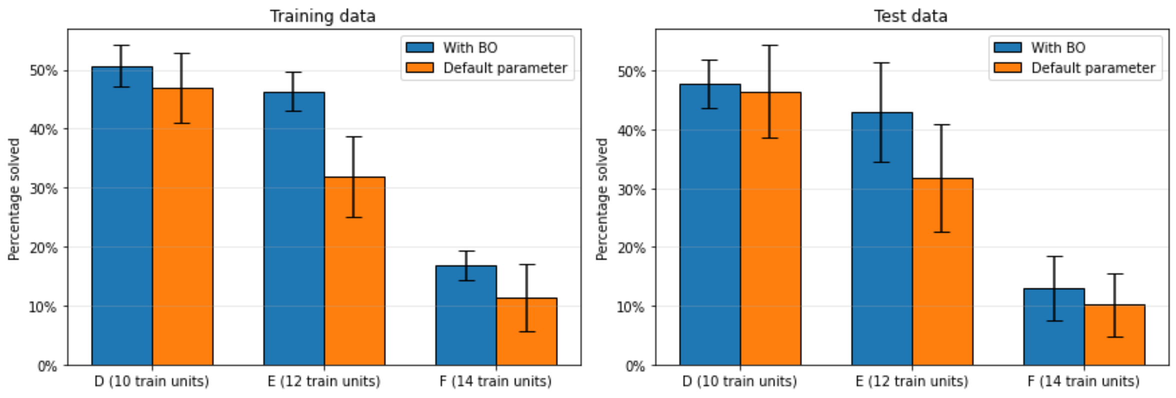


Figure 6.5: Percentage of solved instances for the Idealized location with the original hyperparameter setting and with the best hyperparameter configuration found by the Bayesian Optimization per context. The mean and standard deviations are based on 15 iterations.

therefore, results contain less variability. When looking at the results obtained after testing the best found hyperparameters. The differences in the percentage of solved instances between the two Bayesian Optimization and the default hyperparameters become slightly smaller with experiment D. Besides that, the standard deviation of these optimized hyperparameter configurations is smaller than with the default hyperparameters. However, the differences in test results for experiment D are not significant ($t(21) = 0.55, p = 0.295$). With experiments E and F, the test data shows an increase in the percentage of solved instances when comparing the optimized hyperparameters with the default hyperparameters. The standard deviations of both approaches remain similar in these experiments. Statistically, the test results of experiment E are significantly different when using BO instead of the default hyperparameters ($t(28) = 3.45, p = 0.001$). Contrary, the results of experiment F are not significant ($t(28) = 1.38, p = 0.089$).

If the best performing hyperparameters of the Idealized Location are compared to the default hyperparameters can be seen that the split action has a lower priority, in all the experiments, than it was initially designed with. In almost all the experiments the split action becomes an action with one of the lowest priorities which means that a train is first parked before it is split. However, the Bayesian Optimization creates only for experiment E a significant improvement.

Experiment D and F show both that the move internal action has a substantiated lower priority value if compared with experiment E. This means that when all the trains in a scenario must be combined, the deterministic agent changes its strategy. Within the new strategy trains are being relocated until an action becomes feasible and then the action will be executed. This can be noticed by some of the optimal hyperparameter configurations of experiment E. Some of these hyperparameter configurations have almost similar and highest priority values for the departure, move internal, setback and split actions. This strategy creates in our approach a feasible solution but in real life is such a solution highly inefficient since relocation actions are very expensive.

With this analysis, the results of the locations Kleine Binckhorst and the Idealized Location cannot be compared to each other because the complexity of the physical layout differs. However, it is possible to compare the locations in terms of the differences between the BO and the default hyperparameter configuration. The differences between the two approaches are smaller on the location Kleine Binckhorst than the differences on the Idealized Location. This means that the BO can improve the results more on the Idealized Location. The difference in improvements between both locations is a result of the quality of the initial shunting strategy. The results obtained on the location Kleine Binckhorst are much more intuitive for a human planner since a carousel movement can be made, which makes creating optimal deterministic rules easier. When a train arrives at the Kleine Binckhorst, the human planner always tries to follow the same carousel movement. However, within the Idealized Location, this carousel movement cannot be made, and a more complex relocation technique has to be used. This relocation technique is less intuitive for human planners. Therefore, for the Idealized Location, it is much harder for a human planner to formulate a solid shunting strategy and thereby optimal deterministic rules. As a result, the dynamics of the initial deterministic rules for the Kleine Binckhorst are closer to the optimum than for the Idealized Location. Consequently, the formulated hypothesis for the location Kleine Binckhorst can be accepted. This means that the BO does not create better results than the default hyperparameter configuration. For the Idealized Location, can the hypothesis be partly rejected, which means that BO can develop better solutions for experiment E than the default hyperparameter setting, the other improvements can be seen but are not statistically significant.

6.1.4 Discussion on the hyperparameter optimization

Now that all the experiments are executed, can the best hyperparameter configuration be determined per location and experiment. The experiments that do not consider any contextual information for both locations result in a poor generalization from best-obtained hyperparameters on the training data to the test data. The poor generalization occurs more often when considering contextual hyperparameter optimization techniques (Char et al., 2019). This often happens when the surrogate model is not able to detect variations in the complexity of contextual information (Char et al., 2019). Moreover, when comparing the test results of the BO with the default hyperparameter configuration can be noticed that the performance is almost equal to each other, for both locations. Consequently, can the default hyperparameter configuration be considered the best hyperparameter when not considering any contextual information.

When considering contextual information, the poor generalization from the training to the test instances reduces substantiated since the complexity is logically separated through the contextual information. However, the BO could not generate a significant improvement on the training and test data for the location Kleine Binckhorst. After analyzing the results per iteration it became apparent that a similar performance could be obtained by turning the carousel movement in the other direction. However, this did not lead to an improved result and therefore is the default hyperparameter configuration the best configuration that can be applied for experiments A, B and C. The priority values of the optimal hyperparameter configurations are visible in Table 6.1.

The results of the Idealized Location showed a better generalization from the training to the test data. More specifically, the contextual approach finds an improved hyperparameter configuration for experiment E. Experiment E, only consisted of a significant difference between the optimized and the default hyperparameters for both the training and test data. Therefore, the best hyperparameter configuration is selected by taking the best scoring hyperparameter considering the training and test results. This hyperparameter configuration had the highest number of feasible scenarios on the training data and the one to highest number of feasible scenarios on the test data. Experiments D and F significantly improved the training data, but this improvement was not significant on the test data. Therefore, the best hyperparameter configuration for these experiments is the default configuration. A summary of the optimal hyperparameter configuration per experiment is given (Table 6.1).

In conclusion, the Bayesian Optimization approach is not able to improve the hyperparameter configuration for carousel shunting yards. This is because it is more intuitively for the human planners to design an optimal hyperparameter configuration due to the carousel movements. For the shuffleboard locations this intuitive advantage diminishes and therefore it is less likely that the hyperparameters are close to the optimum. However, only a significant difference can be made when all the trains in a scenario must leave the shunting yard combined.

Table 6.1: Optimal hyperparameter configuration for all experiments

Experiments	Connect	Depart	Move entering	Move Internal	Service	Setback	Split	Wait
A, B and C	280	400	200	100	360	200	380	100
D, F	320	400	300	100	NA	260	320	100
E	309.24	274.10	301.04	148.32	NA	202.80	106.10	40.92

6.2 Baseline results for handling stochastic variables

The baseline results for handling the non-stationary variables consist of the results of the deterministic agent without any additions (plain heuristics), and the deterministic agent improved with the Monte Carlo Tree Search (MCTS). For Kleine Binckhorst, an additional enhancement combines the deterministic agent with MCTS and an improved relocation heuristic. This section will consider the hyperparameter configuration of the deterministic agent it is originally designed with for all experiments, A till F, formulated in chapter 5. The baselines are formulated through an experiment that analyzes 1,000 scenarios per approach and per experiment. Besides that, both locations are considered throughout this baseline formulation. The results are visible in Table 6.2.

Table 6.2: Baseline results, percentage solved instance measured by analyzing 1,000 instances per experiment.

Percentage feasible experiments	**A**	**B**	**C**	**D**	**E**	**F**
Solved with plain heuristics	85.60%	51.80%	55.90%	**49.20%**	32.60%	14.50%
Solved with MCTS	86.50%	56.90%	58.00%	47.50%	**32.60%**	**15.30%**
Solved with MCTS and relocation	**91.70%**	**63.40%**	**60.70%**			

The table shows that the enhancement methods improve the performance on all the scenarios for Kleine Binckhorst. For the Idealized Location only scenario F, which consists of 14 train units, shows an improvement. To give a more thoroughly understanding of the results, other performance metrics can be observed. These are the average computation time per scenario and the average number of relocation actions executed. Both are displayed in Table 6.3. As can be observed in the table, the computation times of the experiments executed on the location Kleine Binckhorst (experiment A, B, and C) are much higher than the computation times of the experiments related to the Idealized Location (experiments D,

Table 6.3: Explanatory performance metrics that will be used to give an indication about the quality of the solutions. actions

Average computation time per experiment (sec)	**A**	**B**	**C**	**D**	**E**	**F**
Solved with plain heuristics	28.45	28.17	30.25	4.76	8.48	10.45
Solved with MCTS	56.31	56.31	87.13	37.66	35.42	49.05
Solved with MCTS and relocation	66.99	118.01	93.48			
Total number of relocations per experiment	**A**	**B**	**C**	**D**	**E**	**F**
Solved with plain heuristics	2,535	10,485	10,971	39,466	16,052	31,517
Solved with MCTS	2,916	7,384	8,712	37,829	19,588	30,932
Solved with MCTS and relocation	12,183	17,933	19,236			

E, and F). This is because the number of available actions for the Kleine Binckhorst is much higher than in the Idealized Location since there are more tracks and the service activities are enabled in those scenarios. Consequently, the deterministic agent must assign more priority values, which is time-consuming since it needs to be repeated for every train when changing from state. This makes it complex to compare the computation times of both locations to each other. Therefore, the computation times are compared with experiments on similar locations. When using MCTS, the computation times increase a lot. This is because the MCTS must be repeated every time something changes on the tracks. Therefore, the computation time increases relatively the most, with the experiments on the Idealized Location. Besides that, the roll out phase of the MCTS for the Idealized Location finds a solution less easily, and therefore, it needs more iterations before convergence. If the MCTS and the improved relocation are added to the deterministic agent of the Kleine Binckhorst, the computation time increases a little more because the relocation action can trigger more MCTS iterations. Consequently, the MCTS is a costly addition to the deterministic agent.

The number of relocation actions can say something about the solutions' quality because relocation actions are expensive to have in a shunting schedule. This is because valuable time is required to execute the relocation since the train driver must walk to the other side of the train. These relocation actions are not an objective but rather an explanatory performance metric that will give more insights into the approach. As can be seen, the number of relocation actions increases over the experiments for the location Kleine Binckhorst. For this location, more train units require a more complex movement strategy and, therefore, more relocation actions. Consequently, the MCTS increases the quality of the solution by making fewer relocation actions, but this comes at a cost, which is the computation time of the scenarios. When adding the MCTS and the advanced relocation strategy together, the number of relocation actions increases, which means that the quality of the solution decreases. But nevertheless, the number of solved instances increases even further. The results of the Idealized location are slightly different. Here the total number of relocation actions is much higher than with the other location. This makes sense since the shuffleboard design of the shunting yard requires more relocation actions before a suitable shunting plan can be made. Besides that, it is expected that the MCTS results in fewer relocation actions because the quality of these actions is higher. Moreover, it was expected that the increase in the number of train units was connected to an increasing number of relocation actions. This is because more train units should create more relocation actions. However, this cannot be observed from the obtained results.

Next, an in-depth analysis is executed to look at what scenarios are solved through which enhancement method. This in-depth analysis is done to give a broader understanding of the

Table 6.4: Unique solved scenarios and newly created sequential model to solve the Train Unit Shunting Problem

Experiment	**A**	**B**	**C**	**D**	**E**	**F**
Solved with plain heuristics	21	59	54	25	43	13
Solved with MCTS	10	51	87	8	43	21
Solved with MCTS and relocation	34	99	91			
Total	65	209	232	33	86	34

created approaches. With this analysis, it became apparent that most of the enhancement approaches were solving similar scenarios. However, each enhancement method also had a part that was uniquely solved by that specific approach. To indicate how often this occurred, a table is made that consists of the uniquely solved instances per approach (Table 6.4). Subsequently, this means that these enhancement methods can solve more scenarios if grouped. Therefore, another experiment is created that uses the different models sequentially to solve the scenarios. If a method is not able to create a feasible solution, another approach will be tried. The performance of these subsequent methods is displayed in Table 6.5.

If the performance of the sequential model is compared with the benchmark algorithm, the performance of experiment A comes close to the performance of the benchmark algorithm. Experiments B and C obtain a performance increase of more than 20% compared with the baseline model, but still, the benchmark algorithm solves 17 or 18% more scenarios. The performance increase of experiments D, E, and F can be considered moderate and does not increase to a level that can approach the performance of the benchmark algorithm. The main takeaway is that the experiments with more train units benefit more from the sequential model than experiments with a small number of train units. The computation times for the sequential model are similar to the computation times for the enhancement that contained the MCTS and the improved relocation strategy, namely 31.68 sec, 118.01 sec, 113.46 sec, 19.23 sec, 56.12 sec, 77.66 sec for experiment A till F, respectively.

Table 6.5: Performance of the sequential model compared with the benchmark algorithm.

Experiment	**A**	**B**	**C**	**D**	**E**	**F**
Solved with sequential model	97.50%	79.90%	81.40 %	50.00%	36.90%	16.60%
HIP (benchmark algorithm)	**99.90%**	**98.70%**	**99.20%**	**99.90%**	**98.00%**	**74.10%**

Based on these obtained results can be concluded that the enhancement methods have the desired effects, especially for the Kleine Binckhorst. The added value for the location Kleine Binckhorst is much higher than for the Idealized Location. This is because the number of possible end tracks is higher for the Kleine Binckhorst which means that it is easier for the MCTS enhancement to find a good solution. Since there are more simulation possibilities. Besides that, due to the large number of considered tracks on the Kleine Binckhorst is the deterministic agent better able to recover from earlier made mistakes. Which means that the MCTS enhancement can be applied again. This is often not possible for the Idealized Location due to the low number of tracks.

6.2.1 Violation analysis

The obtained results can be analyzed more thoroughly by analyzing the violations that caused the infeasible actions. A summary of these violations per experiment can be found in Figure 6.6. This figure shows that the violations per location differ, for the location Kleine Binckhorst (experiments A, B, and C) the largest violation is *train cannot leave*, after that a large part of the scenarios fail because of the violation *composition is not present*, the other violations are negligible. Experiment A for the Kleine Binckhorst consist of a total of 15 train units and 15 outgoing trains. This means that there is no combination required when solving experiment A. With 16 train units, the number of splits and combinations become two and one, respectively. With 17 train units the number of splits and combinations becomes one and one, respectively. This means that the total number of splits and combinations is higher with experiment B than with experiment C. Consequently, an increase with violation *composition not present* is visible. However, it is interesting to see that only the splitting causes an increase in the violation, *composition not present*, when it is executed together with a combination.

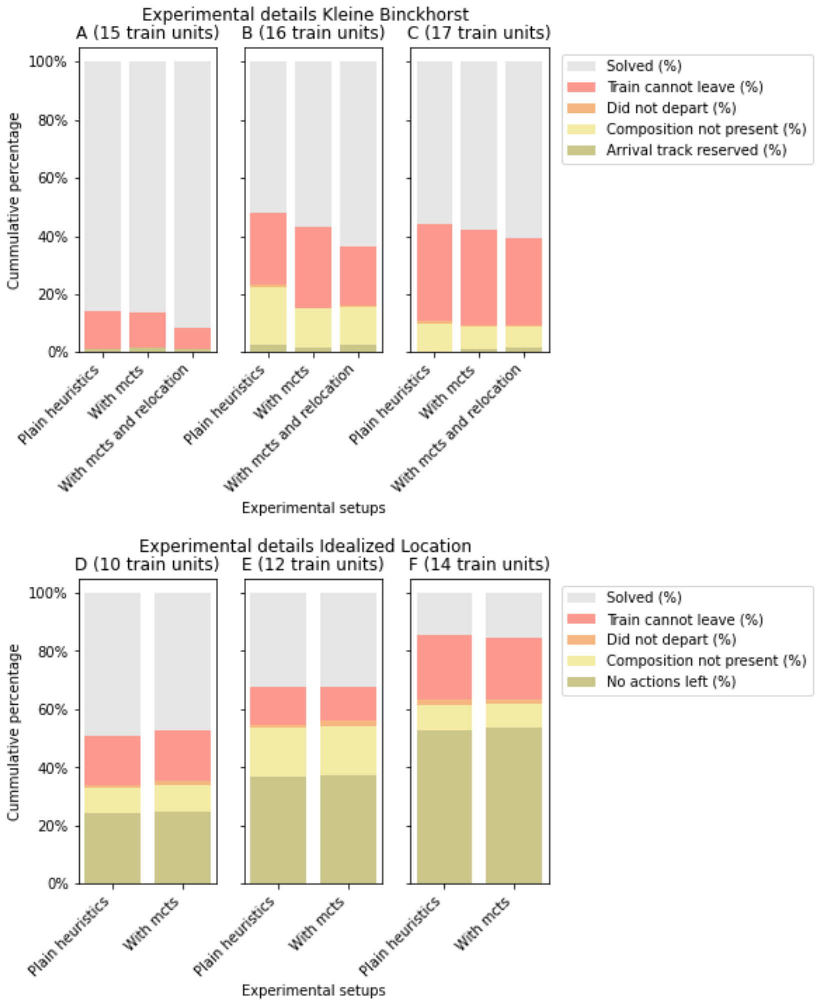


Figure 6.6: Representation of the violations per location, scenario composition and the enhancement method if used.

This can be concluded since experiment A requires two splits no combination and there is no violation with *composition not present* is present. Contrary, the violation *train cannot leave* increases throughout the different experiments. This indicates that the movements on the tracks become more complex when solving for more train units. Formulated otherwise, the deterministic agent is, to a lesser extent, able to recover from earlier mistakes when the number of train units increases.

The Idealized Location (experiments D, E, and F) does not suffer from the violation *arrival track reserved* while it consists of another violation, namely, *no actions left.* This violation means that the simulation model has put itself in a deadlock position, and it has no possible actions left. After an in-depth analysis, it was thought that a part of this violation could be allocated to the fact that the relocation track is full. Then the trains cannot relocate, which means that the simulation model quickly runs out of viable actions. To confirm this statement, the length of the relocation track of the idealized location is temporally extended from 500 to 3,000 meters. With this extension, all the arriving trains fit on the relocation track and therefore this error should occur less often. For this additional experiment, the plain heuristics are used with the experiment that suffers the most from this violation, namely experiment F. This experiment is executed by analyzing 1,000 scenarios. When increasing the relocation track to an unrealistic size, the number of feasible instances increases to 20.5%, but the expected results did not occur. Most violations decreased while the violation *no actions left* increased. The violation *train cannot leave* decreased with 6.3%, the violation *composition not present* decreased with 2%, the violation *did not depart* remained the same and remarkably the violation *no actions left* increased with 2.3%. After a more thorough analysis, it became clear that the characteristics of the shunting yard and the scenarios have a dependency on each other and therefore, a combination of multiple variables is causing a violation. We will explain this by formulating a combination of dependencies. The percentage of available track length when all the trains are on the tracks is 47.3% when considering experiment D this increases to 69.4% when considering experiment F. This means that the available space for movements decreases with 20%. This in combination with the increasing number of splits and combinations, and the fact that a maximum of only four trains can make a move (since there are only four different tracks), makes the decision to choose the right action and not get a violation even harder.

Like we mentioned before, with every increasing number of train units in the scenario, the number of splits and combinations increases. However, the number of times a composition is not present decreases when comparing experiments E and F with 12 and 14 train units, respectively. After analyzing some of the results more thoroughly, it became apparent that it is more likely that the simulation stops because it has no actions left than when the composition is not present. This can happen because the error *composition not present* only occurs when the trains are being departed, thus late on the simulation timeline. In contrast to that, the error *no actions left* can occur everywhere in the timeline of the simulation model. Therefore, it is more likely that a scenario becomes infeasible because there are no actions left instead of an infeasible scenario through a composition that is not present.

6.3 Improved hyperparameters and handling stochastic variables

This section will elaborate on the results when combining the improved hyperparameters with the analyzed search strategy. The default hyperparameter configuration is the best configuration for almost all the scenarios, only for experiment E, a significant improvement can be made when choosing other hyperparameters. This section only focuses on the improvement that can be made in experiment E because the results of the default hyperparameters are formulated in the previous section. For these experiments, the baseline experiment is repeated but then with the improved hyperparameter configuration. The values of the optimized hyperparameter configuration is given in Table 6.1. The results of these experiments are given in Table 6.6.

Table 6.6: Experimental results with optimal hyperparameter configuration.

Experiment E	**Feasible**	**Computation time**	**Nr. relocations**
Solved with plain heuristics	46.50%	6.52 sec	14,787
Solved with MCTS	45.70%	66.86 sec	14,912
Solved sequential model	50.70%	49.72 sec	NA

If the results of the plain heuristics are compared with the baseline results, the percentage of feasible instances increases by 13.9%. This means that the optimized hyperparameters contribute to more feasible scenarios. When comparing the results that use the MCTS with the baseline, a difference of 13.1% can be noticed. The MCTS also makes an improvement, but this improvement is smaller. This can be explained by the fact that the hyperparameter optimization is executed with the plain heuristics. Therefore, the resulting hyperparameter configuration is optimized to work best on plain heuristics. When placing the models in a sequence, an improvement of 13.8% can be realized. Therefore, the difference in hyperparameters results in roughly 13% more feasible scenarios when looked over the different experiments. Interesting to see is that the plain heuristics and the MCTS enhancement increase with similar margins.

Another aspect is the computation times. The computation times for the plain heuristics are almost similar when comparing for both hyperparameter configurations but the computation times for the MCTS extension increases by more than 30 seconds. These higher computation times are a result of an increase in demand for additional information. The sequential methods only deviate around the 6 seconds in favor of the optimized hyperparameters. This makes sense since there are more feasible scenarios with the plain heuristics and thereby fewer scenarios for which both methods must be applied. The number of relocation actions decreases which makes sense since the connect action has the highest priority and the split action has the smallest priority. In this way, mostly complete trains are moved instead of train units which results in fewer relocation actions.

A comparison of the violation is made in Figure 6.7. The violations *train cannot leave* and *composition not present* benefit the most from the improved parameters. Especially the violation *composition not present* occurs less often. This behavior is as expected since the connect action has received the highest score (Table 6.1). This means that the improved hyperparameter configuration better connects trains and creates more feasible shunting plans. The violations *no actions left* stayed the same after the hyperparameter optimization.

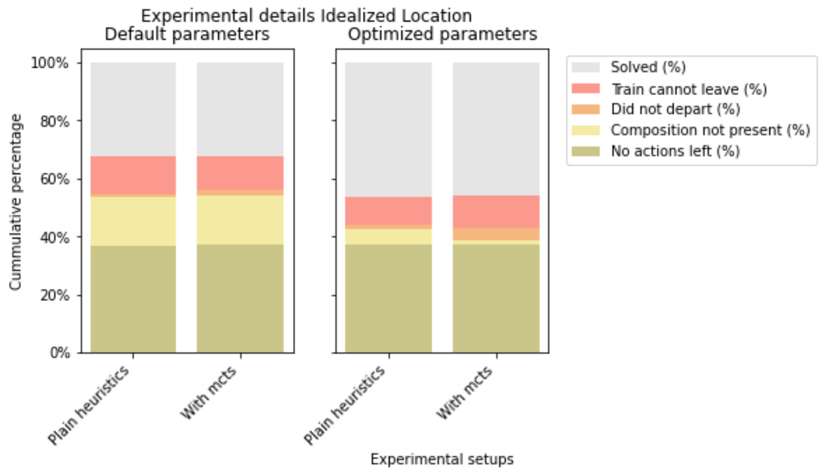


Figure 6.7: Violations of the default and the optimized hyperparameters for experiment E.

6.4 Discussion

Many experiments are executed, which generate lots of results. This section will discuss the findings of the last chapter shortly. In the first paragraph, the results of the hyperparameter optimization are elaborated, the second paragraph will shortly recap the baseline results and the final paragraph will elaborate on the combination of both approaches.

The first part of this section elaborates the results of the experiments that compare the hyperparameters optimized through Bayesian Optimization with the default hyperparameters. These experiments consist of two types of approaches. Namely, one set of experiments that does not consider any contextual information and one set with experiments that does consider contextual information. The contextual information in these experiments is the deviation of scenarios based on among others, the number of train units. The experiments that do not consider any contextual information show a poor generalization from best-obtained hyperparameters on the training data to the unencountered test data. The poor generalization occurs more often when considering contextual hyperparameter optimization techniques, especially when the surrogate model is not able to detect variations in the underlying complexity of the contextual information (Char et al., 2019). Therefore, the poor generalization is reduced by including the contextual information within the experiments. It became apparent that the surrogate model could better generalize the best-found hyperparameter configuration obtained on the training data to the not encountered test data. This means that splitting the scenarios based on contextual information makes the complexity of the scenarios more homogeneous but does not create an improved hyperparameter configuration for all instances. More specifically, the Bayesian Optimization approach is not able to improve the hyperparameter configuration for carousel shunting yards. This is because it is more intuitively for the human planners to design an optimal hyperparameter configuration due to the carousel movements. For the shuffleboard locations this intuitive advantage diminishes and therefore it is less likely that the hyperparameters are close to the optimum. However, the Bayesian Optimization can only create a significant difference when all the trains in a scenario must leave the shunting yard combined.

Then the experiments that use the MCTS to handle the stochasticity and thereby creating more feasible shunting plans. It can be concluded that the feasibility rate improves the most when the shunting yard is a carousel-like shunting yard. The shuffleboard-like shunting yards are not benefiting from the extension with the MCTS. This difference is a result of the number of considered tracks in the MCTS because more tracks give the MCTS more options and also a better possibility to recover from earlier made mistakes. Therefore, the deterministic agent of the carousel-like shunting yard has more room for improvement. After analyzing the lacking number of feasible shunting plans on the shuffleboard location, it became apparent that each solution approach solves a unique set of experiments. Therefore, an additional experiment is created where the different solution approaches are placed in sequence. When placing the models in sequence, the number of feasible shunting plans increases further for both locations and approaches similar feasibility rates as the benchmark algorithm for some experiments. Therefore, there is an increase in feasible shunting plans when using the MCTS, but this comes with costs in the form of increased computation times. Combining the optimized hyperparameters with the MCTS leads to an improvement. This is mainly because the simulation model is better able to combine train units which decreases the violation *composition not present.*

Chapter 7

Conclusion and Recommendations

This research project is a study that focuses on solving the parking-, routing- and service scheduling- subproblems of the Train Unit Shunting Problem (TUSP) for the Dutch Railways. The main objective is to investigate to what extent a framework consisting of hyperparameter optimization and heuristic search approaches can contribute to creating more feasible shunting plans in an online scheduling approach. By doing so, this research ensures increased applicability for the creation of online shunting schedules and increased applicability of hyperparameter optimization techniques. For this research, a literature study is executed. This literature study provided a broad understanding of the approaches that are researched to solve the TUSP. Besides that, insights are created about hyperparameter optimization techniques and heuristic search techniques. Subsequently, these insights are used to design a framework for optimizing the hyperparameters and designing a better search technique that is able to handle stochasticity. In this chapter, the conclusion and subsequently the recommendations are formulated.

7.1 Conclusion

Throughout this project, a simulation model is used to create a shunting plan. This is done by executing the shunting plan online and keeping track of the executed actions. The actions are selected by a deterministic agent that consists of a set of deterministic rules. Each deterministic rule assigns a predefined priority value to an action. When all viable actions are equipped with priority values, the highest valued action is executed in the simulation environment. This process is repeated until all trains in the simulation model have left the shunting yard. Then, the shunting plan is called feasible. In order to cope with the demand to create more feasible shunting plans and to increase the applicability of hyperparameter optimization techniques a framework, of two sequential steps has been designed accordingly. An iterative Bayesian Optimization technique is designed to optimize the priority values that remain fixed when executing the algorithm. Changing the hyperparameters will change the dynamics of the algorithm since an action becomes less or more important than other actions. Within this Bayesian Optimization, the response surface model is modeled through Gaussian Processes and the most promising hyperparameter configurations are chosen through a Gaussian Process Upper Confidence Bound approach. The problem is formulated as a satisfaction problem and in the experiments contextual information is distinguished. The experiments are executed on two types of shunting yards, a carousel shunting yard and a shuffleboard shunting yard. The contextual information distinguished in these experiments is the deviation of scenarios based on, among others, the number of train units. The experiments that do not consider any contextual information show a poor generalization from best-obtained hyperparameters on the training data to the unencountered test data, for both locations. The experiments that consider contextual information show that the surrogate model was better able to generalize the best-found hyperparameter configuration obtained on the training data to the not encountered test data. However, this difference can only be called statistically significant for one experiment that consisted of a single contextual category on the shuffleboard shunting yard. For all other contextual categories, the hyperparameters from the initial design were the best hyperparameters. Therefore can be concluded that splitting the scenarios based on contextual information makes the complexity of the scenarios more homogeneous but does not create an improved hyperparameter configuration for all instances.

The second part of the framework, a Monte Carlo Tree Search (MCTS) heuristic, is designed to handle the stochasticity. Stochasticity can arise when movement actions are approached in a deterministic manner because these actions can depend on the composition of future arrival or departure sequences. Previously, a random choice was made if the deterministic rules were not sufficiently specified. We designed an MCTS to overcome these random choices by using Monte Carlo estimates of feasible parked trains. The parking of trains is called feasible if all the trains can leave to the gateway track unobstructed. The computation times of this MCTS are made efficient by incorporating a bias in the simulation phase. Subsequently, the MCTS is integrated with the existing deterministic rules through a proactive MCTS that gathers new Monte Carlo estimates whenever a change in the dedicated tracks occurs. In this manner, the MCTS always contains the most actual information. The experiments are executed on two different types of shunting yards, a carousel shunting yard and a shuffleboard shunting yard. Besides this MCTS, another relocation strategy is implemented for the carousel-like shunting yard. After the experiments can be concluded that the feasibility rate improves the most when the shunting yard is a carousel-like shunting yard. The shuffleboard-like shunting yards are not benefiting from the extension with the MCTS. This difference is a result of the number of considered tracks in the MCTS because more tracks give the MCTS more options and also a better possibility to recover from earlier made mistakes. Therefore, the deterministic agent of the carousel-like shunting yard has more room for improvement. After analyzing the lacking number of feasible shunting plans on the shuffleboard location it became apparent that each solution approach solves a unique set of experiments. Therefore, an additional experiment is created where the different solution approaches are placed in sequence. When placing the models in sequence, the number of feasible shunting plans increases further for both locations and approaches similar feasibility rates as the benchmark algorithm for some problem instances. Therefore can be concluded that there is an increase in feasible shunting plans when using the MCTS but this comes with a costs in the form of increased computation times.

Overall, the framework presented in this work showed exciting and novel methods for applying hyperparameter optimization techniques and optimizing the number of feasible shunting plans. More specifically, this framework demonstrated that the number of feasible solutions could be optimized by applying a hyperparameter optimization technique and improving the search capabilities of the deterministic agent. An iterative Bayesian Optimization method provided a technique to optimize the dynamics of an algorithm, thereby optimizing the number of feasible shunting plans and increasing the applicability of these optimization techniques. Subsequently, the MCTS introduced a method to handle the stochasticity by creating Monte Carlo estimates of feasible parking positions. These estimates are used to provide the deterministic agent with additional information. By using this framework the number of feasible shunting plans increase substantiated and is approaching the performance of the benchmark algorithm for some problem instances.

7.2 Limitations and recommendations

This research project served as an investigation to what extent an optimization framework could increase the applicability of hyperparameter optimization techniques and the number of feasible shunting plans generated by a set of deterministic rules in a simulation model. Thereby, increasing the understanding and applicability of the online creation of shunting schedules. This work shows a proof of concept with such an optimization framework but due to the limited time frame of the master thesis, some aspects and challenges are not included in the research. Therefore, we advise the NS the following;

1. First, throughout this work, the quality of the shunting plans is not considered, an indication is given through the additional performance metrics, but this is not sufficient. This means that the created shunting plans can consist of an unreasonable number of moves or actions that do not make sense. When these approaches are used for future researches, it is recommended to do additional research that focuses on explaining and, if necessary, improving the quality of the solutions.

2. Secondly, the performance for the Idealized Location is much lower than the benchmark algorithm, even with the best hyperparameters configuration and the Monte Carlo Tree Search. Therefore, we recommend further optimizing the deterministic agent for the Idealized Location with other approaches that might be better suited for shuffleboard locations.

3. Thirdly, when using this optimization framework in future applications is recommended to use the sequential model. This model creates the highest number of feasible shunting schedules on all experiments. When using that model, some speed-up methods might be helpful. Throughout this thesis, a parallel approach is used, but this specification should depend on the implementation details of the future application.

Bibliography

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256. 39

Bardenet, R., Brendel, M., Kégl, B., and Sebag, M. (2013). Collaborative hyperparameter tuning. In *International conference on machine learning*, pages 199–207. PMLR. 15

Barnhoorn, Q. (2020). A multi-agent approach to the train unit shunting problem. 13

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24. 14, 33

Boon, M., , van der Boor, M., van Leeuwaarden, J., Mathijsen, B., van der Pol, J., and Resing, J. (2020). *Stochastic Simulation using Python*. Eindhoven University of Technology, Department of Mathematics and Computer Science. 48

Buhrman, H. and De Wolf, R. (2002). Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43. 34

Char, I., Chung, Y., Neiswanger, W., Kandasamy, K., Nelson, A. O., Boyer, M., Kolemen, E., and Schneider, J. (2019). Offline contextual bayesian optimization. *Advances in Neural Information Processing Systems*, 32:4627–4638. v, 55, 56, 59, 66

Chaslot, G., Saito, J.-T., Bouzy, B., Uiterwijk, J., and Van Den Herik, H. J. (2006). Monte-carlo strategies for computer go. In *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, pages 83–91. 17

Chaslot, G. M. J.-B. C. (2010). *Monte-carlo tree search*. Maastricht University. 16, 18, 37

Chung, Y., Char, I., Neiswanger, W., Kandasamy, K., Nelson, A. O., Boyer, M. D., Kolemen, E., and Schneider, J. (2020). Offline contextual bayesian optimization for nuclear fusion. *arXiv preprint arXiv:2001.01793*. 15

Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer. 17

Eggensperger, K., Lindauer, M., and Hutter, F. (2019). Pitfalls and best practices in algorithm configuration. *Journal of Artificial Intelligence Research*, 64:861–893. xi, 29

Falkner, S., Klein, A., and Hutter, F. (2018). Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pages 1437–1446. PMLR. 46

Forrester, A. I., Sóbester, A., and Keane, A. J. (2007). Multi-fidelity optimization via surrogate modelling. *Proceedings of the royal society a: mathematical, physical and engineering sciences*, 463(2088):3251–3269. 15

Freling, R., Lentink, R. M., Kroon, L. G., and Huisman, D. (2005). Shunting of passenger train units in a railway station. *Transportation Science*, 39(2):261–272. 4, 8

Gallo, G. and Miele, F. D. (2001). Dispatching buses in parking depots. *Transportation Science*, 35(3):322–330. 9

Gelly, S. and Silver, D. (2007). Combining online and offline knowledge in uct. In *Proceedings of the 24th international conference on Machine learning*, pages 273–280. 18, 39

Girolami, M. (2011). *A first course in machine learning*. Chapman and Hall/CRC. 35

Haahr, J., Lusby, R., and Wagenaar, J. (2015). A comparison of optimization methods for solving the depot matching and parking problem. Technical report. 3, 9

Haijema, R., Duin, C., and Van Dijk, N. M. (2006). Train shunting: A practical heuristic inspired by dynamic programming. *Planning in intelligent systems: aspects, motivations, and methods*, pages 437–475. 3, 9

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., et al. (2018). Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32. 3

Huang, D., Allen, T. T., Notz, W. I., and Miller, R. A. (2006). Sequential kriging optimization using multiple-fidelity evaluations. *Structural and Multidisciplinary Optimization*, 32(5):369–382. 15

Hutter, F., Hoos, H., and Leyton-Brown, K. (2013). An evaluation of sequential model-based optimization for expensive blackbox functions. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, pages 1209–1216. 47, 56

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2010). Sequential model-based optimization for general algorithm configuration (extended version). *Technical Report TR-2010–10, University of British Columbia, Computer Science, Tech. Rep.* 14, 15, 33

Jacobsen, P. M. and Pisinger, D. (2011). Train shunting at a workshop area. *Flexible services and manufacturing journal*, 23(2):156–180. xi, 10

Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer. 17, 39

Kroon, L. G., Lentink, R. M., and Schrijver, A. (2008). Shunting of passenger train units: an integrated approach. *Transportation Science*, 42(4):436–449. iii, 3, 9

Kushner, H. J. (1964). A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. 14, 35

Lee, W.-J., Jamshidi, H., and Roijers, D. M. (2020). Deep reinforcement learning for solving train unit shunting problem with interval timing. In *European Dependable Computing Conference*, pages 99–110. Springer. 3, 12, 13, 41

Lentink, R. (2006). *Algorithmic decision support for shunt planning.* Number 73. 9

Lentink, R. M., Fioole, P.-J., Kroon, L. G., and Van't Woudt, C. (2006). Applying operations research techniques to planning of train shunting. *Planning in Intelligent Systems: Aspects, Motivations, and Methods*, pages 415–436. 8

Lindauer, M. and Hutter, F. (2018). Warmstarting of model-based algorithm configuration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32. 45

Močkus, J. (1975). On bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference*, pages 400–404. Springer. 14, 35

Peer, E., Menkovski, V., Zhang, Y., and Lee, W.-J. (2018). Shunting trains with deep reinforcement learning. In *2018 ieee international conference on systems, man, and cybernetics (smc)*, pages 3063–3068. IEEE. xi, 3, 11, 12

Rasmussen, C. E. (2003). Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer. 14, 33

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489. iv, 16

Srinivas, M. and Patnaik, L. M. (1994). Genetic algorithms: A survey. *computer*, 27(6):17–26. 13

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. (2009). Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*. 14, 35

Strassen, V. (1969). Gaussian elimination is not optimal. *Numer. Math*, 13:354–356. 34

Student (1908). The probable error of a mean. *Biometrika*, pages 1–25. 48, 54

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press. xi, 16

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294. 15

Van Den Akker, M., Baarsma, H., Hurink, J., Modelski, M., Jan Paulus, J., Reijnen, I., Roozemond, D., and Schreuder, J. (2008). Shunting passenger trains: getting ready for departure. 3, 10

van den Bogaerdt, P. (2018). Multi-machine scheduling lower bounds using decision diagrams. Master's thesis, TU Delft. 9

van den Broek, R. W. (2016). Train shunting and service scheduling: an integrated local search approach. Master's thesis, University Utrecht. xi, 3, 11, 12, 16, 41, 50

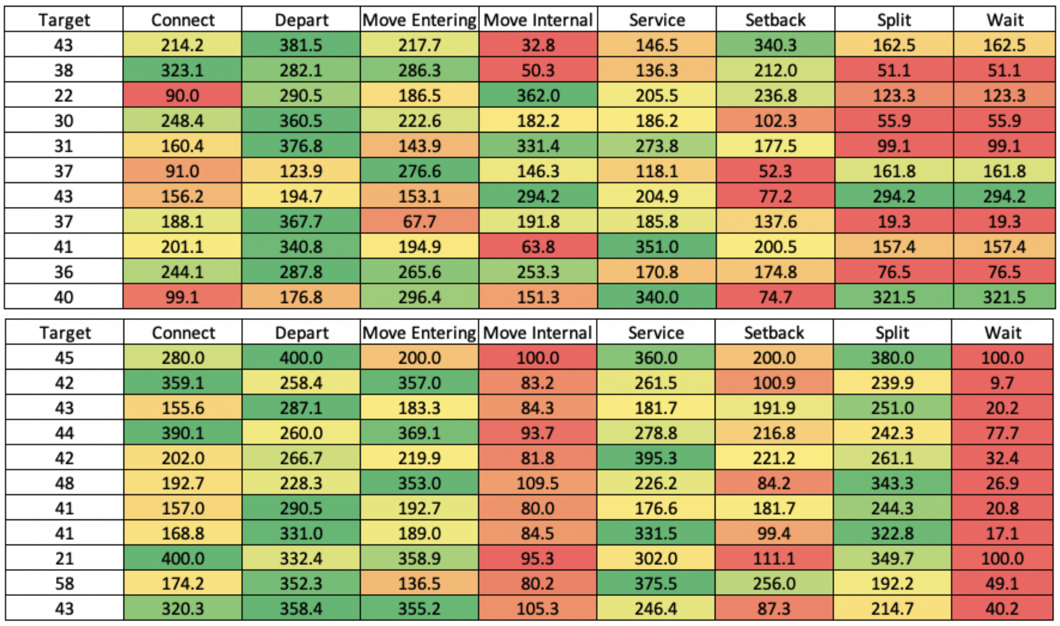
van der Knaap, L. (2021). Contextual hyperparameter optimization for the train unit shunting problem. Master's thesis, Technische Universiteit Deflt. 15

Yogatama, D. and Mann, G. (2014). Efficient transfer learning method for automatic hyperparameter tuning. In *Artificial intelligence and statistics*, pages 1077–1085. PMLR. 15

Appendix A

Detailed results for number of iterations

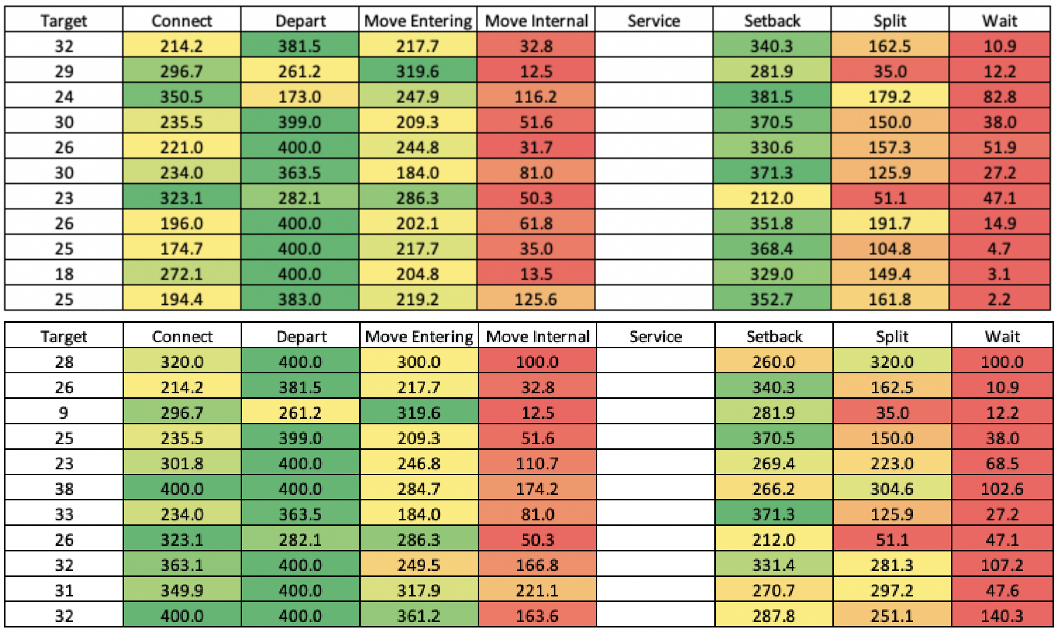
This appendix gives a part of the hyperparameters that are obtained in the experiment to determine number of iterations. Figure A.1 and Figure A.2 give the first eleven hyperparameter sets that did not returned zero feasible scenarios for the location Kleine Binckhorst and the Idealized Location, respectively. Within both figures the upper table represents the hyperparameters found when not considering the warm start approach, the lower table represents the hyperparameters when using the warm start approach. As can be seen when comparing these figures, the Bayesian Optimization is not able to align the hyperparameters of the Kleine Binckhorst such that many feasible solutions can be created, this improves when using the warm start approach. Note that eleven hyperparameters are chosen as reference since only eleven solutions were found that did not returned zero feasible scenarios in the experiment without the warm start approach for the Kleine Binckhorst.



Target	Connect	Depart	Move Entering	Move Internal	Service	Setback	Split	Wait
43	214.2	381.5	217.7	32.8	146.5	340.3	162.5	162.5
38	323.1	282.1	286.3	50.3	136.3	212.0	51.1	51.1
22	90.0	290.5	186.5	362.0	205.5	236.8	123.3	123.3
30	248.4	360.5	222.6	182.2	186.2	102.3	55.9	55.9
31	160.4	376.8	143.9	331.4	273.8	177.5	99.1	99.1
37	91.0	123.9	276.6	146.3	118.1	52.3	161.8	161.8
43	156.2	194.7	153.1	294.2	204.9	77.2	294.2	294.2
37	188.1	367.7	67.7	191.8	185.8	137.6	19.3	19.3
41	201.1	340.8	194.9	63.8	351.0	200.5	157.4	157.4
36	244.1	287.8	265.6	253.3	170.8	174.8	76.5	76.5
40	99.1	176.8	296.4	151.3	340.0	74.7	321.5	321.5

Target	Connect	Depart	Move Entering	Move Internal	Service	Setback	Split	Wait
45	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0
42	359.1	258.4	357.0	83.2	261.5	100.9	239.9	9.7
43	155.6	287.1	183.3	84.3	181.7	191.9	251.0	20.2
44	390.1	260.0	369.1	93.7	278.8	216.8	242.3	77.7
42	202.0	266.7	219.9	81.8	395.3	221.2	261.1	32.4
48	192.7	228.3	353.0	109.5	226.2	84.2	343.3	26.9
41	157.0	290.5	192.7	80.0	176.6	181.7	244.3	20.8
41	168.8	331.0	189.0	84.5	331.5	99.4	322.8	17.1
21	400.0	332.4	358.9	95.3	302.0	111.1	349.7	100.0
58	174.2	352.3	136.5	80.2	375.5	256.0	192.2	49.1
43	320.3	358.4	355.2	105.3	246.4	87.3	214.7	40.2

Figure A.1: The hyperparameter sets for the Kleine Binckhorst of the first eleven solutions that did not returned zero feasible scenarios when not using the warm start approach (upper) and while using the warm start approach (lower). The colors represent the underlying priority where green and red are the actions with the highest and lowest priority, respectively.



Target	Connect	Depart	Move Entering	Move Internal	Service	Setback	Split	Wait
32	214.2	381.5	217.7	32.8		340.3	162.5	10.9
29	296.7	261.2	319.6	12.5		281.9	35.0	12.2
24	350.5	173.0	247.9	116.2		381.5	179.2	82.8
30	235.5	399.0	209.3	51.6		370.5	150.0	38.0
26	221.0	400.0	244.8	31.7		330.6	157.3	51.9
30	234.0	363.5	184.0	81.0		371.3	125.9	27.2
23	323.1	282.1	286.3	50.3		212.0	51.1	47.1
26	196.0	400.0	202.1	61.8		351.8	191.7	14.9
25	174.7	400.0	217.7	35.0		368.4	104.8	4.7
18	272.1	400.0	204.8	13.5		329.0	149.4	3.1
25	194.4	383.0	219.2	125.6		352.7	161.8	2.2

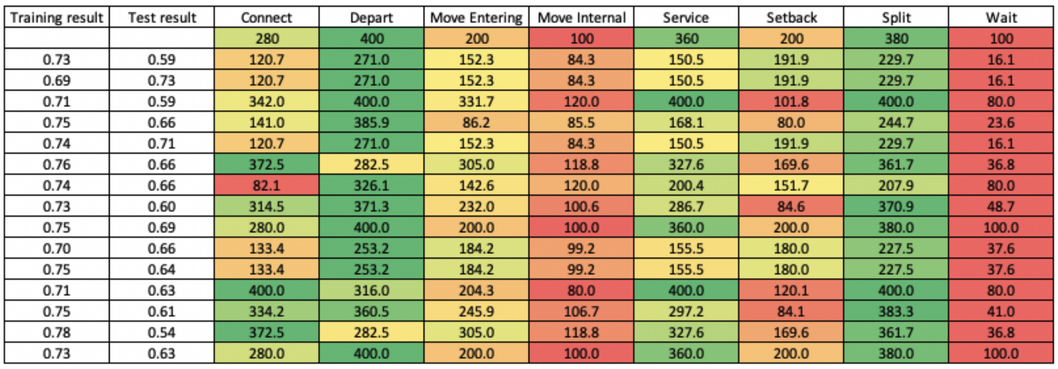
Target	Connect	Depart	Move Entering	Move Internal	Service	Setback	Split	Wait
28	320.0	400.0	300.0	100.0		260.0	320.0	100.0
26	214.2	381.5	217.7	32.8		340.3	162.5	10.9
9	296.7	261.2	319.6	12.5		281.9	35.0	12.2
25	235.5	399.0	209.3	51.6		370.5	150.0	38.0
23	301.8	400.0	246.8	110.7		269.4	223.0	68.5
38	400.0	400.0	284.7	174.2		266.2	304.6	102.6
33	234.0	363.5	184.0	81.0		371.3	125.9	27.2
26	323.1	282.1	286.3	50.3		212.0	51.1	47.1
32	363.1	400.0	249.5	166.8		331.4	281.3	107.2
31	349.9	400.0	317.9	221.1		270.7	297.2	47.6
32	400.0	400.0	361.2	163.6		287.8	251.1	140.3

Figure A.2: The hyperparameter sets for the Idealized Location of the first eleven solutions that did not returned zero feasible scenarios when not using the warm start approach (upper) and while using the warm start approach (lower). The colors represent the underlying priority where green and red are the actions with the highest and lowest priority, respectively.

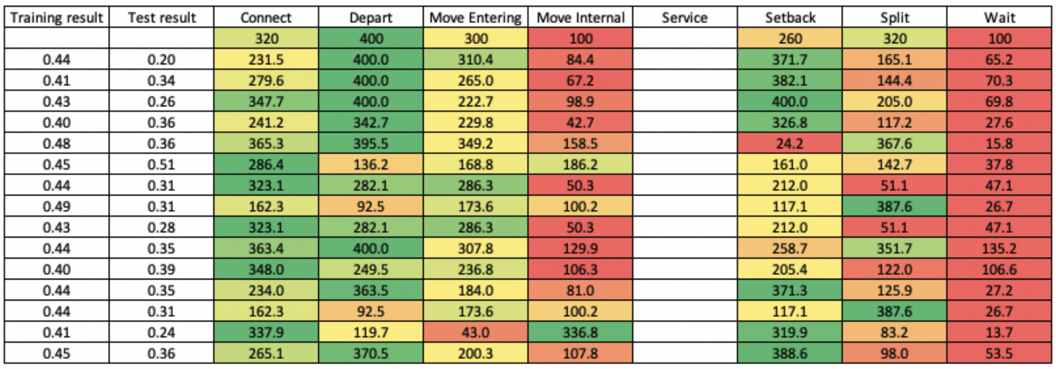
Appendix B

Detailed results without contextual information

This appendix gives the hyperparameters that are obtained within the experiments that do not consider any contextual information. The first row of the tables in Figure B.1 gives the default hyperparameters and as can be seen does the obtained hyperparameter configurations not deviate much from that priority. Therefore can be concluded that the overfitting is a cause of the inefficient sampling strategy.



Training result	Test result	Connect	Depart	Move Entering	Move Internal	Service	Setback	Split	Wait
		280	400	200	100	360	200	380	100
0.73	0.59	120.7	271.0	152.3	84.3	150.5	191.9	229.7	16.1
0.69	0.73	120.7	271.0	152.3	84.3	150.5	191.9	229.7	16.1
0.71	0.59	342.0	400.0	331.7	120.0	400.0	101.8	400.0	80.0
0.75	0.66	141.0	385.9	86.2	85.5	168.1	80.0	244.7	23.6
0.74	0.71	120.7	271.0	152.3	84.3	150.5	191.9	229.7	16.1
0.76	0.66	372.5	282.5	305.0	118.8	327.6	169.6	361.7	36.8
0.74	0.66	82.1	326.1	142.6	120.0	200.4	151.7	207.9	80.0
0.73	0.60	314.5	371.3	232.0	100.6	286.7	84.6	370.9	48.7
0.75	0.69	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0
0.70	0.66	133.4	253.2	184.2	99.2	155.5	180.0	227.5	37.6
0.75	0.64	133.4	253.2	184.2	99.2	155.5	180.0	227.5	37.6
0.71	0.63	400.0	316.0	204.3	80.0	400.0	120.1	400.0	80.0
0.75	0.61	334.2	360.5	245.9	106.7	297.2	84.1	383.3	41.0
0.78	0.54	372.5	282.5	305.0	118.8	327.6	169.6	361.7	36.8
0.73	0.63	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0



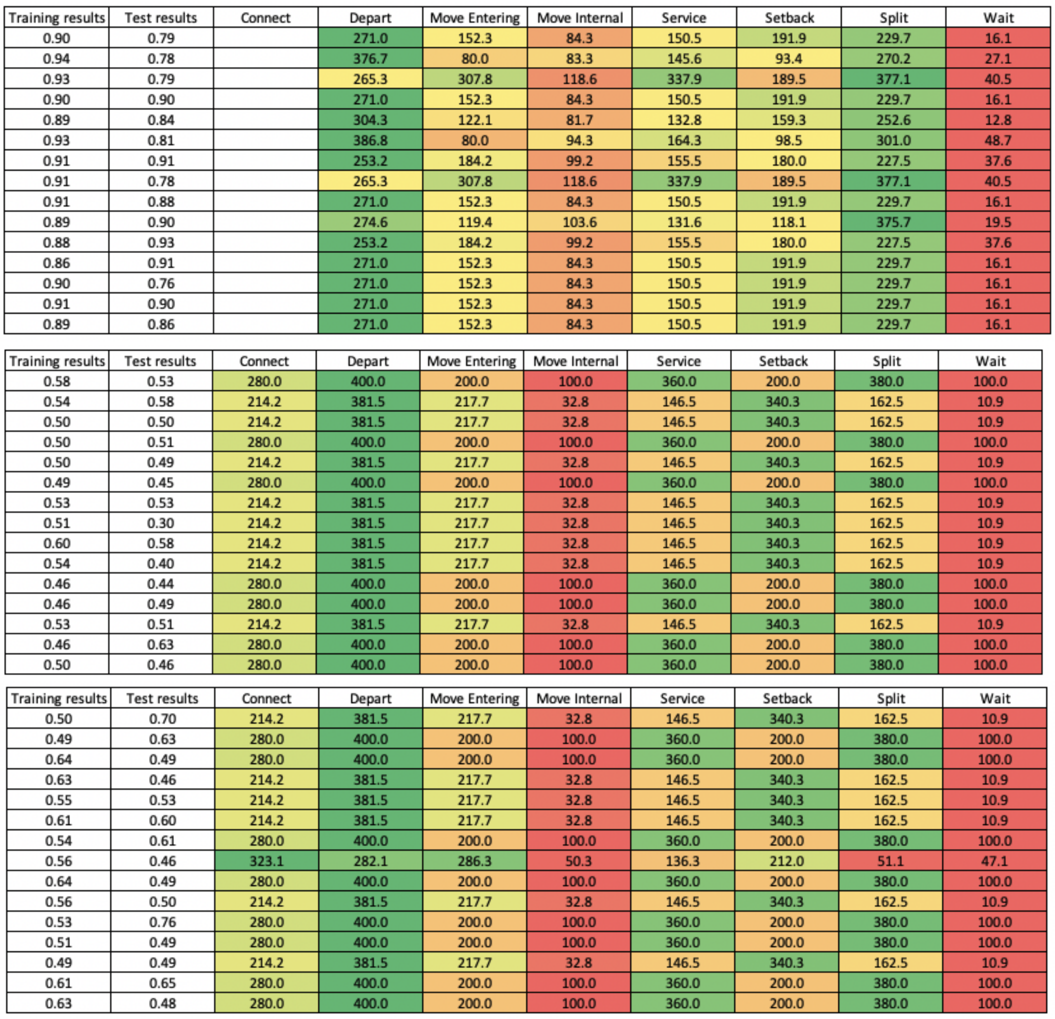
Training result	Test result	Connect	Depart	Move Entering	Move Internal	Service	Setback	Split	Wait
		320	400	300	100		260	320	100
0.44	0.20	231.5	400.0	310.4	84.4		371.7	165.1	65.2
0.41	0.34	279.6	400.0	265.0	67.2		382.1	144.4	70.3
0.43	0.26	347.7	400.0	222.7	98.9		400.0	205.0	69.8
0.40	0.36	241.2	342.7	229.8	42.7		326.8	117.2	27.6
0.48	0.36	365.3	395.5	349.2	158.5		24.2	367.6	15.8
0.45	0.51	286.4	136.2	168.8	186.2		161.0	142.7	37.8
0.44	0.31	323.1	282.1	286.3	50.3		212.0	51.1	47.1
0.49	0.31	162.3	92.5	173.6	100.2		117.1	387.6	26.7
0.43	0.28	323.1	282.1	286.3	50.3		212.0	51.1	47.1
0.44	0.35	363.4	400.0	307.8	129.9		258.7	351.7	135.2
0.40	0.39	348.0	249.5	236.8	106.3		205.4	122.0	106.6
0.44	0.35	234.0	363.5	184.0	81.0		371.3	125.9	27.2
0.44	0.31	162.3	92.5	173.6	100.2		117.1	387.6	26.7
0.41	0.24	337.9	119.7	43.0	336.8		319.9	83.2	13.7
0.45	0.36	265.1	370.5	200.3	107.8		388.6	98.0	53.5

Figure B.1: The best performing hyperparameters obtained when executing the experiments without considering any contextual information for the Kleine Binckhorst (upper) and the Idealized Location (lower). The colors represent the underlying priority where green and red are the actions with the highest and lowest priority, respectively. Each row depicts a Bayesian Optimization sequence and the first row depicts the default hyperparameters as reference.

Appendix C

Detailed results with contextual information

This appendix gives the obtained hyperparameters when the contextual information is considered per experiment. Figure C.1 and Figure C.2 represent the obtained hyperparameters for the Kleine Binckhorst and the Idealized Location, respectively. Figure C.1 clearly shows that there are two hyperparameter configurations that obtain similar training and test results for the location Kleine Binckhorst. One of these configurations is the default hyperparameter configuration. The improved hyperparameters are not able to create a significant improvement on the test data for the Kleine Binckhorst. Figure C.2 shows that the default hyperparameters are not the best configuration for the Idealized Location since every iteration finds an improved hyperparameter configuration. However, the performance is only significantly better with experiment E.

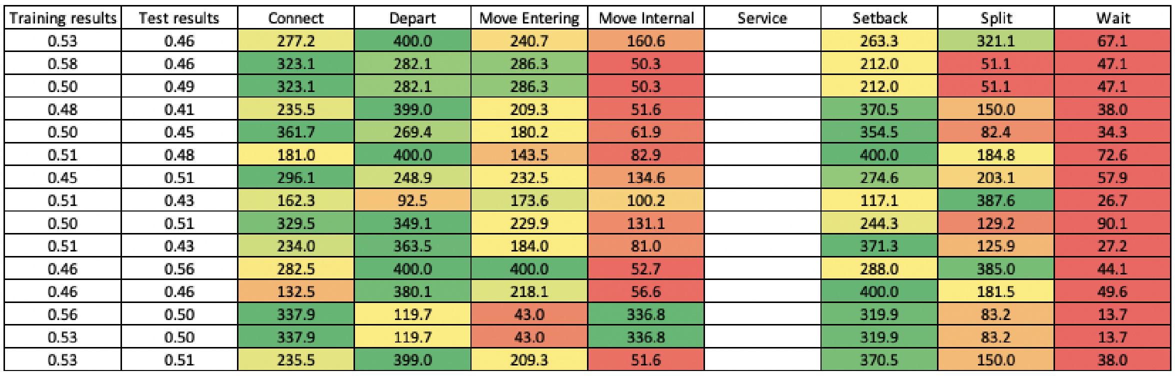


Training results	Test results	Connect	Depart	Move Entering	Move Internal	Service	Setback	Split	Wait
0.90	0.79		271.0	152.3	84.3	150.5	191.9	229.7	16.1
0.94	0.78		376.7	80.0	83.3	145.6	93.4	270.2	27.1
0.93	0.79		265.3	307.8	118.6	337.9	189.5	377.1	40.5
0.90	0.90		271.0	152.3	84.3	150.5	191.9	229.7	16.1
0.89	0.84		304.3	122.1	81.7	132.8	159.3	252.6	12.8
0.93	0.81		386.8	80.0	94.3	164.3	98.5	301.0	48.7
0.91	0.91		253.2	184.2	99.2	155.5	180.0	227.5	37.6
0.91	0.78		265.3	307.8	118.6	337.9	189.5	377.1	40.5
0.91	0.88		271.0	152.3	84.3	150.5	191.9	229.7	16.1
0.89	0.90		274.6	119.4	103.6	131.6	118.1	375.7	19.5
0.88	0.93		253.2	184.2	99.2	155.5	180.0	227.5	37.6
0.86	0.91		271.0	152.3	84.3	150.5	191.9	229.7	16.1
0.90	0.76		271.0	152.3	84.3	150.5	191.9	229.7	16.1
0.91	0.90		271.0	152.3	84.3	150.5	191.9	229.7	16.1
0.89	0.86		271.0	152.3	84.3	150.5	191.9	229.7	16.1

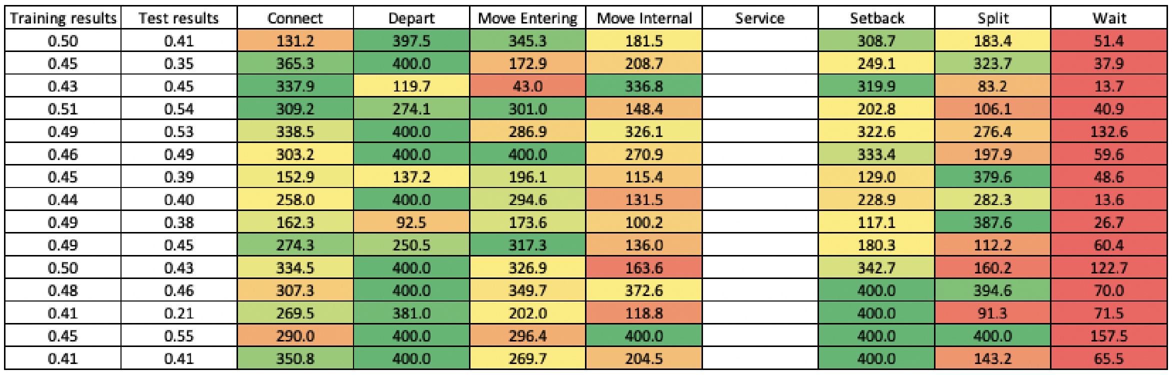
Training results	Test results	Connect	Depart	Move Entering	Move Internal	Service	Setback	Split	Wait
0.58	0.53	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0
0.54	0.58	214.2	381.5	217.7	32.8	146.5	340.3	162.5	10.9
0.50	0.50	214.2	381.5	217.7	32.8	146.5	340.3	162.5	10.9
0.50	0.51	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0
0.50	0.49	214.2	381.5	217.7	32.8	146.5	340.3	162.5	10.9
0.49	0.45	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0
0.53	0.53	214.2	381.5	217.7	32.8	146.5	340.3	162.5	10.9
0.51	0.30	214.2	381.5	217.7	32.8	146.5	340.3	162.5	10.9
0.60	0.58	214.2	381.5	217.7	32.8	146.5	340.3	162.5	10.9
0.54	0.40	214.2	381.5	217.7	32.8	146.5	340.3	162.5	10.9
0.46	0.44	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0
0.46	0.49	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0
0.53	0.51	214.2	381.5	217.7	32.8	146.5	340.3	162.5	10.9
0.46	0.63	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0
0.50	0.46	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0

Training results	Test results	Connect	Depart	Move Entering	Move Internal	Service	Setback	Split	Wait
0.50	0.70	214.2	381.5	217.7	32.8	146.5	340.3	162.5	10.9
0.49	0.63	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0
0.64	0.49	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0
0.63	0.46	214.2	381.5	217.7	32.8	146.5	340.3	162.5	10.9
0.55	0.53	214.2	381.5	217.7	32.8	146.5	340.3	162.5	10.9
0.61	0.60	214.2	381.5	217.7	32.8	146.5	340.3	162.5	10.9
0.54	0.61	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0
0.56	0.46	323.1	282.1	286.3	50.3	136.3	212.0	51.1	47.1
0.64	0.49	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0
0.56	0.50	214.2	381.5	217.7	32.8	146.5	340.3	162.5	10.9
0.53	0.76	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0
0.51	0.49	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0
0.49	0.49	214.2	381.5	217.7	32.8	146.5	340.3	162.5	10.9
0.61	0.65	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0
0.63	0.48	280.0	400.0	200.0	100.0	360.0	200.0	380.0	100.0

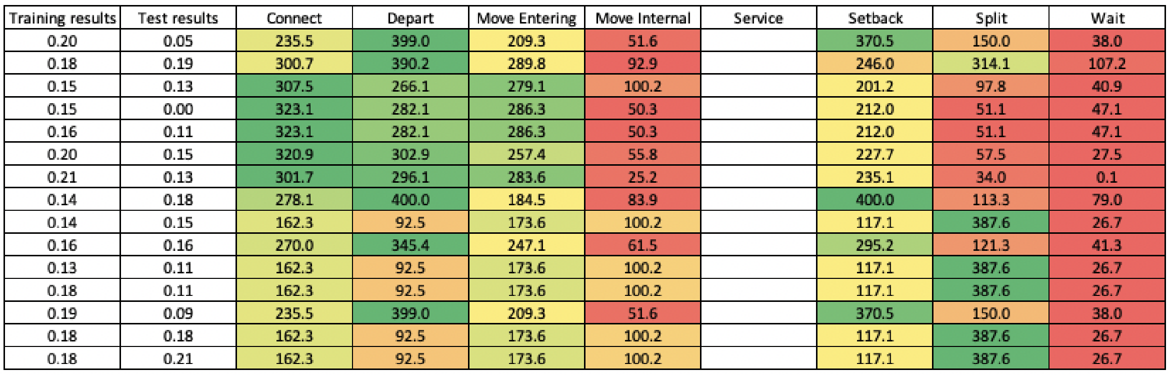
Figure C.1: The hyperparameter configurations obtained when executing the experiments that consider contextual information for the location Kleine Binckhorst. The upper, middle and lowest tables represent the found hyperparameters for the scenarios A, B and C. The colors represent the underlying priority where green and red are the actions with the highest and lowest priority per set, respectively. Each row depicts a Bayesian Optimization sequence.



Training results	Test results	Connect	Depart	Move Entering	Move Internal	Service	Setback	Split	Wait
0.53	0.46	277.2	400.0	240.7	160.6		263.3	321.1	67.1
0.58	0.46	323.1	282.1	286.3	50.3		212.0	51.1	47.1
0.50	0.49	323.1	282.1	286.3	50.3		212.0	51.1	47.1
0.48	0.41	235.5	399.0	209.3	51.6		370.5	150.0	38.0
0.50	0.45	361.7	269.4	180.2	61.9		354.5	82.4	34.3
0.51	0.48	181.0	400.0	143.5	82.9		400.0	184.8	72.6
0.45	0.51	296.1	248.9	232.5	134.6		274.6	203.1	57.9
0.51	0.43	162.3	92.5	173.6	100.2		117.1	387.6	26.7
0.50	0.51	329.5	349.1	229.9	131.1		244.3	129.2	90.1
0.51	0.43	234.0	363.5	184.0	81.0		371.3	125.9	27.2
0.46	0.56	282.5	400.0	400.0	52.7		288.0	385.0	44.1
0.46	0.46	132.5	380.1	218.1	56.6		400.0	181.5	49.6
0.56	0.50	337.9	119.7	43.0	336.8		319.9	83.2	13.7
0.53	0.50	337.9	119.7	43.0	336.8		319.9	83.2	13.7
0.53	0.51	235.5	399.0	209.3	51.6		370.5	150.0	38.0



Training results	Test results	Connect	Depart	Move Entering	Move Internal	Service	Setback	Split	Wait
0.50	0.41	131.2	397.5	345.3	181.5		308.7	183.4	51.4
0.45	0.35	365.3	400.0	172.9	208.7		249.1	323.7	37.9
0.43	0.45	337.9	119.7	43.0	336.8		319.9	83.2	13.7
0.51	0.54	309.2	274.1	301.0	148.4		202.8	106.1	40.9
0.49	0.53	338.5	400.0	286.9	326.1		322.6	276.4	132.6
0.46	0.49	303.2	400.0	400.0	270.9		333.4	197.9	59.6
0.45	0.39	152.9	137.2	196.1	115.4		129.0	379.6	48.6
0.44	0.40	258.0	400.0	294.6	131.5		228.9	282.3	13.6
0.49	0.38	162.3	92.5	173.6	100.2		117.1	387.6	26.7
0.49	0.45	274.3	250.5	317.3	136.0		180.3	112.2	60.4
0.50	0.43	334.5	400.0	326.9	163.6		342.7	160.2	122.7
0.48	0.46	307.3	400.0	349.7	372.6		400.0	394.6	70.0
0.41	0.21	269.5	381.0	202.0	118.8		400.0	91.3	71.5
0.45	0.55	290.0	400.0	296.4	400.0		400.0	400.0	157.5
0.41	0.41	350.8	400.0	269.7	204.5		400.0	143.2	65.5



Training results	Test results	Connect	Depart	Move Entering	Move Internal	Service	Setback	Split	Wait
0.20	0.05	235.5	399.0	209.3	51.6		370.5	150.0	38.0
0.18	0.19	300.7	390.2	289.8	92.9		246.0	314.1	107.2
0.15	0.13	307.5	266.1	279.1	100.2		201.2	97.8	40.9
0.15	0.00	323.1	282.1	286.3	50.3		212.0	51.1	47.1
0.16	0.11	323.1	282.1	286.3	50.3		212.0	51.1	47.1
0.20	0.15	320.9	302.9	257.4	55.8		227.7	57.5	27.5
0.21	0.13	301.7	296.1	283.6	25.2		235.1	34.0	0.1
0.14	0.18	278.1	400.0	184.5	83.9		400.0	113.3	79.0
0.14	0.15	162.3	92.5	173.6	100.2		117.1	387.6	26.7
0.16	0.16	270.0	345.4	247.1	61.5		295.2	121.3	41.3
0.13	0.11	162.3	92.5	173.6	100.2		117.1	387.6	26.7
0.18	0.11	162.3	92.5	173.6	100.2		117.1	387.6	26.7
0.19	0.09	235.5	399.0	209.3	51.6		370.5	150.0	38.0
0.18	0.18	162.3	92.5	173.6	100.2		117.1	387.6	26.7
0.18	0.21	162.3	92.5	173.6	100.2		117.1	387.6	26.7

Figure C.2: The hyperparameter configurations obtained when executing the experiments that consider contextual information for the Idealized Location. The upper, middle and lowest tables represent the found hyperparameters for the scenarios D, E and F. The colors represent the underlying priority where green and red are the actions with the highest and lowest priority per set, respectively. Each row depicts a Bayesian Optimization sequence