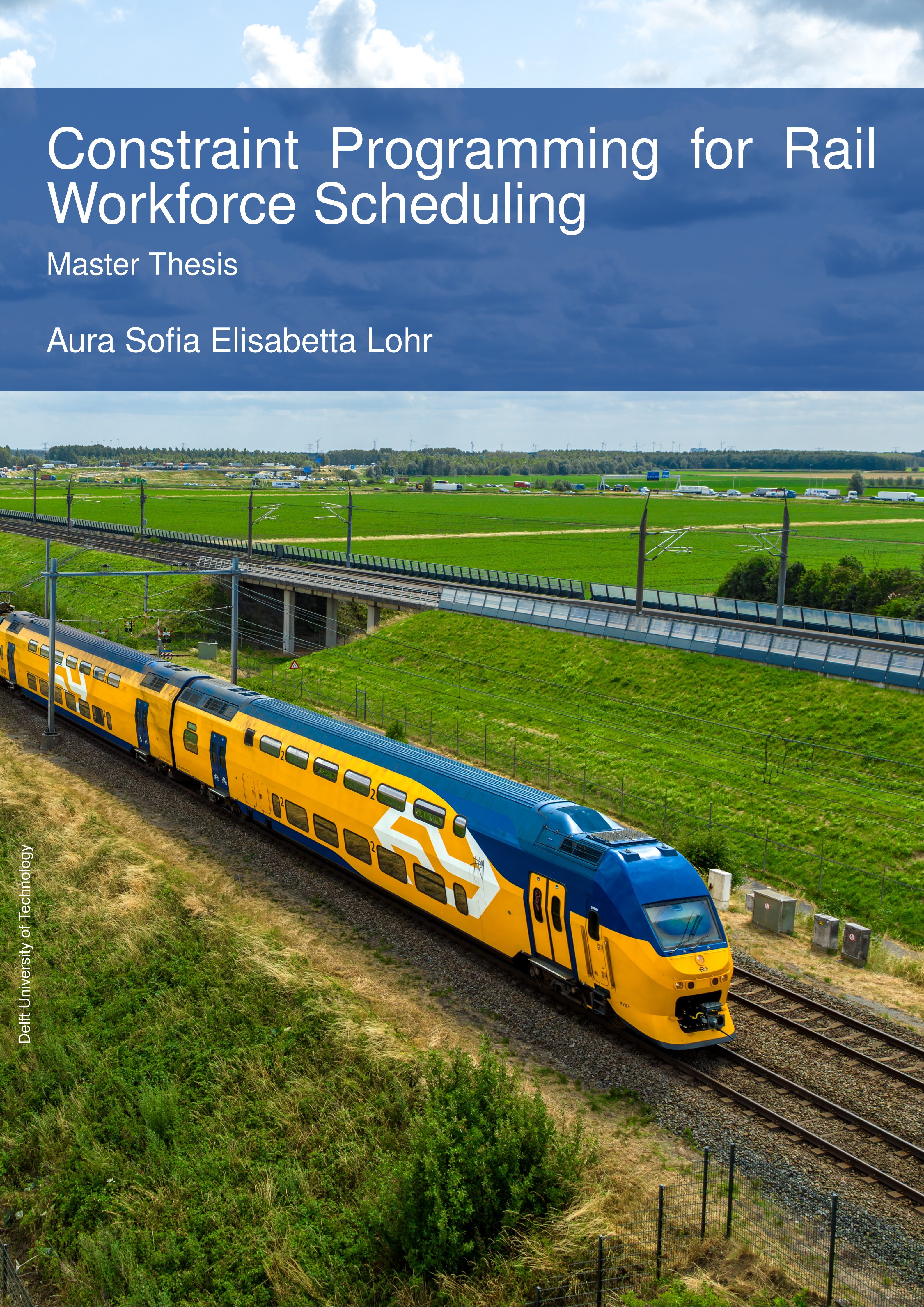
Constraint Programming for Rail Workforce Scheduling

Master Thesis

Aura Sofia Elisabetta Lohr



Delft University of Technology

Constraint Programming for Rail Workforce Scheduling

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Aura Sofia Elisabetta Lohr

Algorithmics Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Nederlandse Spoorwegen
Laan van Puntenburg 100
Utrecht, the Netherlands
www.ns.nl

Cover picture by Kiwi Film.

Constraint Programming for Rail Workforce Scheduling

Author:	Aura Sofia Elisabetta Lohr
Student ID:	6082173

Abstract

As the relevance of rail transport continues to grow, efficient operational planning becomes increasingly critical. This is also evident at stations, where unused trains must be moved to the yard and returned to the platform when needed. To achieve this, available drivers must be assigned to trains in a way that respects the time constraints of their shifts. This thesis explores the approach of combining the shunt routing problem (SRP), where times for routes of shunt trains are calculated, with the drivers' problem, which deals with creating feasible driver schedules. The resulting Constraint Programming (CP) model is based on an existing CP model for the SRP. This combined model is tested on four stations in the Netherlands and solves the problem in under 200 seconds for the Vlissingen and Enkhuizen stations, while higher runtimes for Enschede and Amersfoort are observed. Experiments with an alternative model structure highlight how the linking of the train and driver variables influences runtime significantly. Several variable and value selection strategies are examined on multiple variables, improving performance for certain instances. Furthermore, the analysis reveals that runtime is highly dependent on the station and instance characteristics. By combining the SRP with the drivers' problem, this thesis delivers a functional CP model applicable across all stations and contributes to a deeper understanding of the integration of the two problems.

Thesis Committee:

Chair:	Dr. Emir Demirović, Faculty EEMCS, TU Delft
University supervisor:	Maarten Flippo, Faculty EEMCS, TU Delft
Company supervisor:	Dr. Jord Boeijink, Nederlandse Spoorwegen
Committee Member:	Dr. Christoph Lofi, Faculty EEMCS, TU Delft

Preface

These two years in Delft made my life and mind richer in a myriad of ways and I am forever grateful for the things this place showed me.

I want to thank my supervisors, I felt very well supervised with all of you. Thank you Professor Demirović, even when times were stressful you always found time to check in with me and I really appreciate that. Thank you Maarten for all the good advice, but also all the fun talks. Thank you Jord for always making time for me and the interesting chats in Dutch/Italian/German/English. I felt very comfortable and happy working at NS.

Thank you to all of my housemates, past and present, talking to you guys after a full day in the library always makes my day infinitely better. Thank you Dani for turning even the most mundane moments into lots of fun and making life feel like a true adventure. Thank you Lilli for always being there for me and your brilliant ability to understand me. Thank you to all of the friends that I made along the way, including Marin, Matthijs, Radu, Emma, Andreea, and every fun moment from lunch breaks to karaoke! Also thanks to those far away: Thank you to my friends from Frankfurt, from school to uni!

Thank you to my sisters Chichi and Lollo, you are my everything, thank you for encouraging me in my journey and being there for me. Thank you Mamma for being my biggest fan, always cheering for me, and all of the support throughout the years. Thank you to my father for always believing in me and telling me that you are proud of me when you were still able to. This is also for you, Opa Eisenbahn. I wish I could tell you about this.

Aura Sofia Elisabetta Lohr
Delft, the Netherlands
September 5, 2025

Contents

Chapter 1

Introduction

As individuals become more aware of their carbon footprint, passenger train traffic in Europe is growing. For example, the opening of a new line between Paris and Berlin in December 2024 was so successful that the French and German rail operators, SNCF and DB, decided to expand the line a month after it opened [21]. This momentum is unlikely to diminish: by promoting and investing in rail transport, the EU aims to double high-speed rail traffic by 2030 and triple it by 2050 [9].

This influx of passenger traffic requires rigorous planning to minimize disruptions and delays. For example, in addition to planning the routes of the trains between different stations and cities, the routes of the trains within a station have to be planned meticulously as well. Trains that are not used need to be stored in a yard, and leave the platform as soon as another train might need it. Without a plan for the routes to and from the platforms, trains run the risk of using the same track at the same time, which can cause delays or, even worse, an accident. The creation of this plan for how trains move through the station is called the Shunt Routing Problem (SRP). It is part of the Train Unit Shunting Problem (TUSP), which deals with assigning inbound train units to outbound train units, finding a track to park the train in the yard, scheduling maintenance tasks, and routing the trains to and from the yards. After or during the creation of a shunt plan, the number of drivers needed to execute the routes of the trains and their schedules are calculated.

In the past, the calculation of a shunt plan and the drivers' schedules were done manually. However, with the increase in passenger train traffic, planning is becoming more and more complex, creating the need for an automated planning system to support the planners. This led Nederlandse Spoorwegen (NS), the main passenger railway operator in the Netherlands, to develop the HIP (Hybride Integrale Planmethode), an algorithm that generates a solution for the TUSP at their stations. The algorithm works by creating an initial solution, which is then used as input to a local search, producing a feasible solution. This initial solution is generated step by step, solving problems similar to the above-mentioned subproblems of the TUSP.

Since the choice of the initial solution is relevant for the quality of the subsequent local search, ways to optimize this solution and accelerate performance are explored. For the shunt routing part of the initial solution, Gincheva [10] created a Constraint Programming (CP) model. The model decides which yard to park each train in and calculates the times and routes the trains take to and from the yard.

So far, the drivers' schedules were computed in the local search, after an initial solution for the times and routes of the trains had been calculated. However, the solution to the SRP affects the schedules of the drivers who have to execute the movements of the trains through the station. Therefore, computing the drivers' schedules while the routes to the yards are being calculated can be useful for several reasons, one of which is illustrated in the following example.

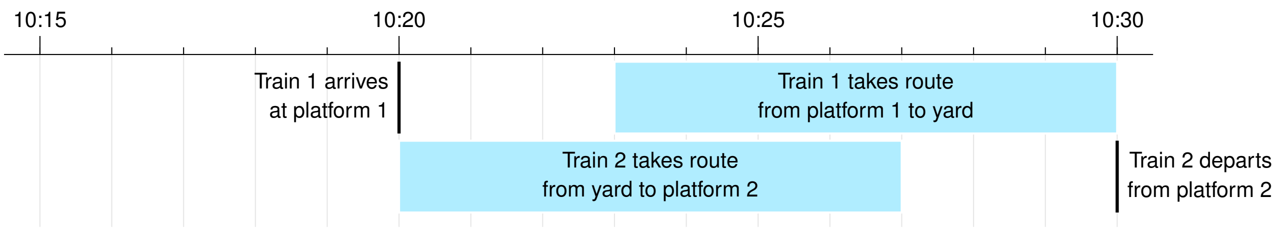


Figure 1.1: Timeline of possible feasible solution of the example with the SRP, a blue block depicts one operation that needs to be executed by one driver

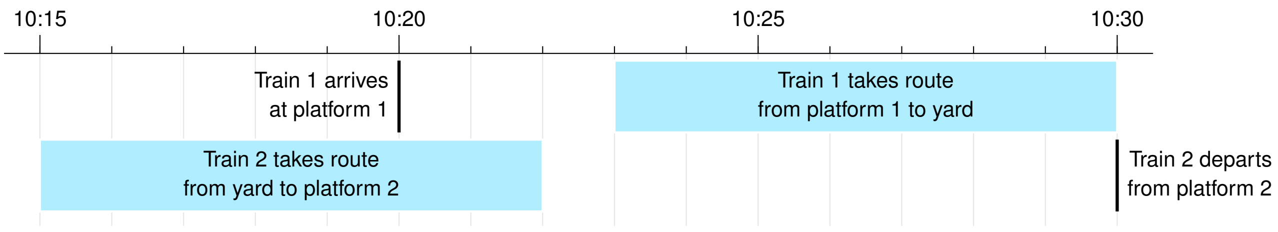


Figure 1.2: Timeline of possible feasible solution of the example with the SRP in combination with the drivers' problem, a blue block depicts an operation that needs to be executed by one driver

In this example, two operations need to be executed at a station: Train 1 arrives at 10:20 on platform 1, and train 2 departs at 10:30 on platform 2. The arriving train 1 must be moved to the yard after three minutes or more, and the departing train 2 must reach platform 2 at least three minutes before the departure. Platform 1 is free from 10:15 to 10:30, platform 2 is free from 10:15 to 10:35.

The SRP could compute the following feasible solution, depicted in Figure 1.1: From 10:20-10:27, train 2 will be brought from the yard to platform 2. From 10:23-10:30, train 1 takes a route from platform 1 to the yard. To execute this plan, two drivers are needed, as two tasks overlap in time. If only one driver were available, the solution would be infeasible.

Computing the SRP in combination with the drivers' problem however could yield this result if only one driver is available: As platform 2 is empty from 10:15, train 2 takes its route from the yard to platform 2 at 10:15-10:22. Train 1 takes its route from platform 1 to the yard at 10:23-10:30. Now, the result depicted in Figure 1.2 is feasible, because the one available driver can execute both movements.

Without taking into account the available drivers, the solution found by the current model could turn out to be infeasible in the local search. By already including the drivers' problem in this step, the solution given to the local search would not need to be modified with respect to the drivers' schedules, and less computation could be required. Additionally, there is another advantage of combining the SRP with the drivers' problem: When the train schedule is changed, it is necessary to check if the available drivers are still able to execute all moves of trains. If the whole HIP would be run again, it would be considerably time-consuming (around one hour for Amersfoort and more than 10 for Utrecht), but just solving the isolated subproblems could keep computation time to a minimum. In order to exploit these benefits, this thesis will deal with the extension of Gincheva's model for the SRP to include the drivers' problem. The drivers' problem will compute the schedules of the available drivers to cover all routes and tasks. This new model, which solves the SRP and the drivers' problem simultaneously, will be implemented in Minizinc. The following main question to be answered is:

How can the drivers' problem be implemented inside the existing CP model for the SRP?

Within this, the following subquestions are addressed:

- To what extent does the addition of the drivers' problem influence the performance of the routing problem?
- How do different solvers or different solving techniques impact the performance of the model?
- Are there certain attributes of a station or instance that influence the performance of the model?

This thesis has the following structure: After a literature review of some relevant papers in the field and a rundown of the algorithms used at NS, relevant terms are defined in the preliminary section. The description of the problem follows, with a detailed formal definition of the new combined model and hypotheses for the following experiments. In the experiments section, the performance of the model is analyzed and compared to Gincheva's model. Modifications of the model are examined, and value and variable selection strategies to improve runtime are tested. The last section concludes the thesis and discusses the main results and future work.

Chapter 2

Related Work

2.1 TUSP at NS

In his PhD thesis, Lentink [17] investigated the TUSP. He split the problem as mentioned earlier into four subproblems: The train matching problem (TMP) matches the train units that arrive to the train units that depart. Some trains arrive and finish their route, others start their route and some get split or need an additional train unit.

After deciding on the matching, the train units that will not depart from that station yet need to be parked somewhere. With the track assignment problem (TAP) it is decided where train units go when they finished their route and do not yet commence the next one.

When moving from the platforms to their shunt tracks (their 'parking spots'), the train units could come in conflict with each other. The shunt routing problem (SRP) tries to minimize these conflicts by finding optimal shunting routes.

Finally, while being parked the trains also need to be cleaned and checked for maintenance, which is planned by maintenance scheduling. The shunt unit cleaning problem (SUCP) for example, belonging to maintenance scheduling, optimizes the cleaning of the trains.

To solve the SRP, an extended A* algorithm and 2-OPT are used. However, sometimes it could result in a train not being assigned a route. Then, the planner would manually need to change some variables.

In 2017, Constraint Programming (CP) was used for the first time to solve parts of the TUSP, although the SRP is not included in this initial application [12].

At that time, NS used a tool named OPG to solve all TUSP subproblems. However, the computation time was so high that it was not an advantage in comparison to using human planners. One identified problem was the decomposition of the problem. When decomposing the problem, it becomes easier to solve as the subproblems are smaller than the big problem. Nonetheless, it is hard to achieve good results with this method as there is no interaction between the different subproblems. To address these issues, van den Broek [4] subsequently developed a local search algorithm to solve the whole TUSP problem. The local search will be deployed at NS to plan the schedule of a station.

After the local search was introduced, there was still a need to reduce computation time. Wattel [22] explored the use of a CP model to generate a solution for the shunt routing problem which could be used as input to the local search. The CP model was made for the Eindhoven train station. By using the CP model, a speed-up of the computation time could be reached.

In 2024 Gincheva [10] then created a general CP model for the initial solution of the SRP that can, in principle, be used at any station. The resulting solutions are conflict-free solutions for the SRP. Gincheva presents two models, the second one being an improved version of the first model. Experiments were carried out with two real-life instances from the Amersfoort and Enkhuizen train stations. Both models have a runtime of less than one second for the Amersfoort instance. Furthermore, when additional routes were added to the Amersfoort instance, the second model still had a runtime of 18 seconds. Enkhuizen resulted in a slower computation time than Amersfoort. Finally, the different solvers were compared, resulting in an exceptionally fast computation with Gecode in comparison to Google OR-Tools. The thesis also raised questions. It is unclear why the Enkhuizen station's computation is slow or how the model performs on different stations. As the model was never connected to the local search, it leads to the question of how this new model influences the local search's performance.

2.2 Crew Scheduling Assignment

While reviewing 123 articles about railway crew scheduling, putting focus on papers published since 2000, Heil et al. [15] observe that integrated approaches such as planning crew schedules alongside the rolling stock have not been studied widely. This is proposed as a future research opportunity.

Van Broek et al. [5] also point out the inconvenience of computing the schedules of the drivers and personnel separate from the shunt plan. This is why they extend the existing local search algorithm [6] to include the assignment of the schedules. Additionally to two heuristics which are added together to the local search, a dynamic program is presented which allows drivers to be passengers in order to minimize their walking time between their activities. The integrated approach for TUSP combined with the dynamic programming resulted in solving 97% of tested instances in a valid amount of time.

In a survey, Castillo et al. [20] attempt to generalize the common features of the scheduling of workforces that have to complete activities in different locations, the Workforce Scheduling and Routing Problem (WSRP). Bourreau et al. [2] go a step further and use a constraint programming-based decomposition method to solve a generalization of the WSRP. The generalization includes temporal dependencies, different levels of workers' skills and the customer and workers' quality of service. The constraint programming-based decomposition consists of three steps, a column generation, a heuristic to maximize the number of coordination constraints and a trip reconstruction, made by the CP solver.

Chen et al. [11] use constraint programming as well as integer programming in a hybrid model to solve the railroad crew problem. The methods are used subsequently to find and assign schedules for the rail crew. However, in their paper, the routes of the trains are already planned.

Next to integer and constraint programming, other approaches are researched as well: Kovacs et al. [16] use an adaptive large neighbourhood search to solve the service technician routing and scheduling problem (STRSP). There, technicians have to complete service tasks that require different levels of skills.

In his thesis, van Nes [18] presents a branch-and-price algorithm for the crew scheduling problem at the yards at NS. The algorithm uses dynamic programming and takes the finished schedules and routes of the trains as input. The algorithm is faster than the one previously used at NS, but only for small instances.

The limited attention to approaches that combine crew scheduling with another problem could demonstrate that combining the two subproblems in this thesis could yield interesting results.

2.3 Freight Scheduling

The problem of scheduling trains in an optimal and feasible way is significant not only for passenger traffic but also for freight traffic. The research on rail freight is therefore relevant as there are many similarities between those two areas. For example, just like passenger trains, freight trains might get split and combined, and need maintenance checks.

When designing a freight station, it is important to accommodate all requirements while still ensuring a good 'flow' through the track groups. Eisold et al. [8] create an optimization model for the dimensioning of track groups using an optimized waiting probability. The objective is to minimize the trains that have to wait before a track group.

Like in passenger train yards, the schedule of a freight train yard needs to be planned rigorously. Preis et al. [19] create an MILP for the scheduling of the resources and track allocations in classification yards. In a classification yard, trains arrive, are disassembled and reassembled again according to their needs, and receive maintenance checks, similar to the split/combine operations and checks before departure. The connections between inbound and outbound trains are not part of the MILP model. The MILP assigns operations and the staff members needed to execute those operations. Additionally, the tracks that the trains take are also being decided. The objective is to minimize the delay of departing trains. Four heuristics are created to obtain an approximated solution. These are also used as an initial solution for the computation of the MILP with CPLEX. In a real-world instance, the methods work well and fast enough to be used for real-time decisions.

Han et al. [14] create a model that solves not only the resource scheduling problem but also the train makeup problem with a hybrid MILP-CP approach. The train makeup problem deals with assigning inbound trains to outbound trains in a yard setting [3], similar to the matching problem in the TUSP. Resource scheduling deals with finding a feasible schedule to complete all operations. Resources are, for example, inspection crew, humping/pullout engine, and hump. To solve the train makeup resource scheduling problem, an MILP is used for the assignment constraints and the CP part is used for the scheduling constraints. For this new model, a logic-based benders decomposition algorithm is created.

The process of scheduling freight trains in terminals is often complicated by the fact that passenger traffic also uses the same tracks. As those two problems are often looked at individually and not intertwined, Haalboom et al. [13] create a two-stage approach to handle rail freight scheduling in combination with a mainline that considers passenger traffic. An initial solution is generated by the first stage. This is used in the second stage where the solution is globally optimized. This approach tries to minimize the difference between the requested paths and the chosen paths, offering an improvement in freight train scheduling.

Chapter 3

Problem Description

3.1 Preliminaries and Term Definitions

A train consists of one or more **train units**. A train unit is a vehicle that contains one or more carriages that are inseparable and can be moved on its own, no locomotive is needed. A reposition of a train is called a **move** for a train, and **drive** for a driver. However, they describe the same thing, that a train is going from one position to another. A **plannable train** describes a train whose routes and times have to be planned in the model, and a **fixed train** refers to a train whose movements are already planned.

3.1.1 Station

A **track** is the underlying structure on which trains are able to drive. The track for on- or off-boarding passengers is called a **platform**. A **switch** allows a train to switch tracks by connecting two tracks. A **yard** is the 'parking spot' for trains when they are not in use or receive maintenance. The entry to a yard is called a **gateway**. The area of the yard and station together is defined as a **node**. A **through train** is a train that has a stop in the station, but only spends time at the platform to offload and onload passengers. A through train is also always a fixed train. With a **reversal**, pictured in Figure 3.1, a train can access another parallel track. It is a time-intensive operation, as the driver has to go to the end of the train to change direction.

3.1.2 Train Compositions

In Gincheva's model [10], a train is defined depending on its current state. A train can be split and/or combined in the yard, or stay as is. A split and combine always happens in a yard, and a split during the first stay at a yard, and a combine during the last stay in a yard upon departure.

An **arrival train** arrives at the platform and finishes its itinerary with passengers there. It exists for at most one movement because after that, it gets split. If the train does not get split, then it is directly a base shunt train and never classified as an arrival train.

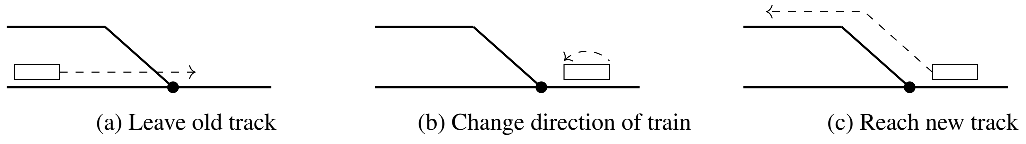

(a) Leave old track (b) Change direction of train (c) Reach new track

Figure 3.1: Reversal

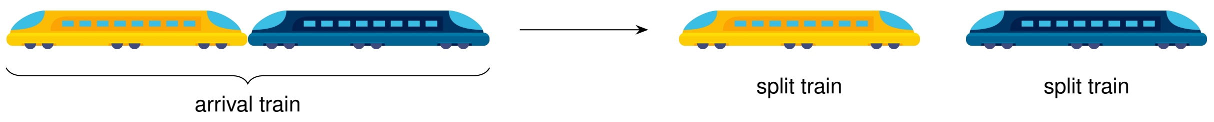


Figure 3.2: Compositions of a train that gets split. It arrives as an arrival train and gets split into split trains, which depart. There is no departure composition. Train images taken from [1].

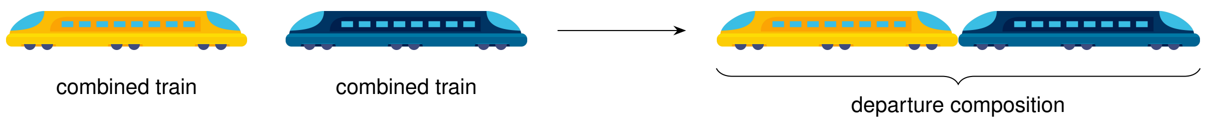


Figure 3.3: Compositions of a train that gets combined. Two train units arrive as combined trains, and they get transformed into a departure composition after being combined. There is no arrival train. Train images taken from [1].

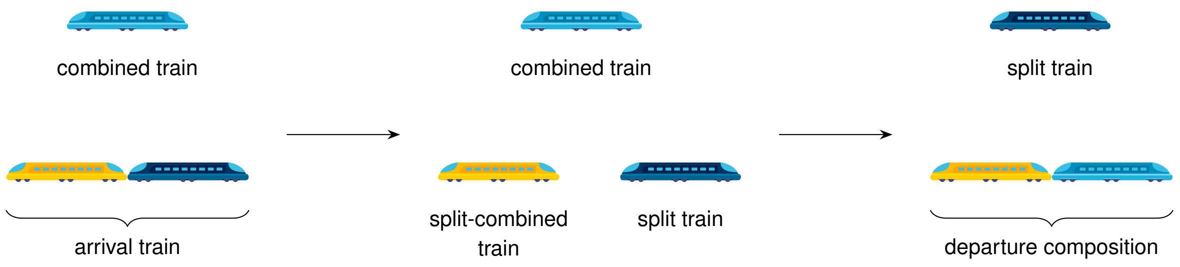


Figure 3.4: Compositions of a train that gets split and combined. Three train units arrive as an arrival train and a combined train, then the arrival train gets split, yielding a split-combined and a split train. The combined and split-combined trains get combined and form a departure composition, while the split train remains a split train until the departure. Train images taken from [1].

A **departure composition** has one movement at most and emerges from a combined or split-combined train. If the train is not combined, the planning will be made directly for the shunt train (base or split).

A **shunt train** is a part of a (or a whole) train that needs to be shunted. There are four types of shunt trains:

- **base shunt train:** This train does not get split or combined. Therefore, it never becomes an arrival or departing train.
- **split shunt train:** This train emerged from a split of an arrival train. As it will not get combined, it will also execute the departure movements. Depicted in Figure 3.3.
- **combined shunt train:** This train is to be combined. It gets transformed into a departure train after the combination. Depicted in Figure 3.2.
- **combined split shunt train:** This train emerges from being split and combined. Depicted in Figure 3.4.

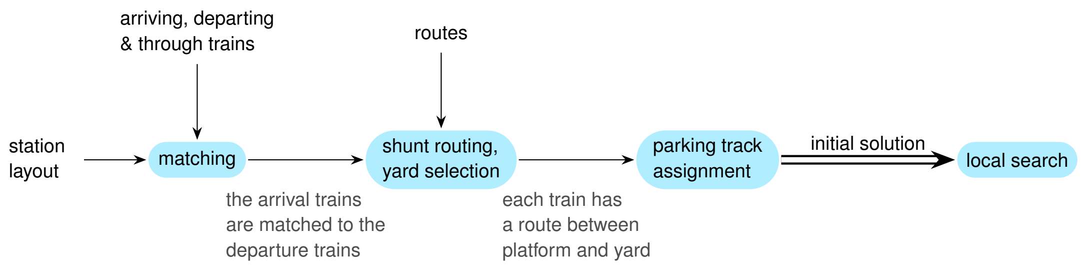

arriving, departing
& through trains
routes
station layout
matching
shunt routing, yard selection
parking track assignment
initial solution
local search
the arrival trains are matched to the departure trains
each train has a route between platform and yard

Figure 3.5: Outline of the HIP

3.2 Problem Description

3.2.1 HIP (Hybride Integrale Planmethode)

The HIP is the algorithm soon to be employed to compute the shunt plan at a station at NS. As mentioned earlier, it uses local search to find a solution. To aid, an initial solution is computed and given as input to the local search. This initial solution is one instance of the problem, and it is mostly infeasible.

The initial solution is computed by splitting the TUSP into its subproblems and then sequentially solving each of them. First, the **matching** of arrival and departure trains is calculated. This matching is given as an input to the next step, which is the **yard selection and shunt routing**. An additional input is the possible routes from each platform to each yard. In yard selection, trains are distributed to the available yards. Additionally, the route each train takes to the yard, the SRP, is computed in this step. Then, the corresponding parking tracks for each train in the yard are assigned in the **parking track assignment**. This step computes the exact positions of the train units within the yard. An overview of these steps can be found in Figure 3.5. After this, the complete initial solution is generated. The result is put into an activity graph, where some details are updated to fit the solutions of the subproblems better together. The final plan graph is then the input for the local search.

3.2.2 CP Model for HIP

To improve the HIP, a CP model for the initial solution for the SRP was designed by Gincheva [10]. This new model differs from the old, where the trains could have a collusion with each other, but not with through traffic. The old model also requires a yard to have the needed facilities available, but as it is not highly significant, it is not implemented in Gincheva's model. The input of the model is the solution of the matching, and it returns the times of the routes taken by the trains.

The model uses time-interval variables. These variables consist of the start time, end time, and duration of a certain event. Using this type of variable is useful as the timing of the routes is crucial to ensure a smooth procedure. Two models are presented in the thesis. The second model is an improved version of the first and has a lower computation time. For the following work with this model, the second version will be used. Therefore, when Gincheva's model is mentioned in this thesis, the second model is meant.

Gincheva made some assumptions when writing the model. It was assumed that through trains that are combined to or split from a plannable train do not occupy the platform when an arriving or departing train is scheduled on the same platform. Moreover, no two trains are allowed to occupy the same section at the same time. The model was tested on instances in which arriving or departing trains that coincide with through trains on the same platform are excluded from the instance. Therefore, to use the model, some preprocessing to remove these unwanted trains is necessary.

To ensure feasibility, some occupied sections were combined into one. This would be the case when their intervals overlapped, leading to two occupied intervals instead of one. Additionally, there is no minimum duration for the trains to be at the platform in order to on- and off-board the passengers. In reality, the trains need to be at the platform for a minimum amount of time. When building upon the model in this thesis, the same assumptions are used.

3.2.3 CP Model in Minizinc

Minizinc is a language for constraint modelling. Using Minizinc allows solving one constraint model with many different solvers such as Google OR-Tools, Gecode, Chuffed, and more. Gincheva created a Minizinc model for the SRP. However, Gincheva's model can only be used for the Enkhuizen instance, as the splits and combines are not implemented, and the Enkhuizen instance only contains base shunt trains. Additionally, the model does not take into account when a train departs after the planning time, which occasionally leads to infeasibility.

In order to use the model for further research, the following changes have been made:

Adding all train types

As the model only worked with base shunt trains, the start and end locations of a train were always known. For example, the first movement always began at a platform at the arrival time of the train. With the addition of combined, split, and split-combined trains, a train could also start its journey at a yard, for example, and have an unknown start time, which will be decided during solving the whole problem.

For every combined train, the final position is set to be equal to the first position of the departure composition, and the first stay of the departure composition has to start after or with the start of the combined train. For every split train, the endpoint of the last move of the arrival train is set equal to the first position of the split train. The start of the stay is set to be equal to or after the start of the last stay of the arrival train. For split-combined trains, both of the above-mentioned constraints were implemented. Moreover, constraints that deal with the arrival or departure of a train were modified, as not every train has an arrival or departure.

Implementation of trains departing the next day

If two or more trains depart after the planning horizon from the same platform, the model will yield an infeasibility. This is because the departing time is set to the maximum number, and two or more trains are set to depart at this time. As these trains are planned in the next planning horizon, they should not be assigned to a route. However, to make sure that the train is parked at the right yard (in case of a combine or split with another train), the yard should be assigned according to the matching.

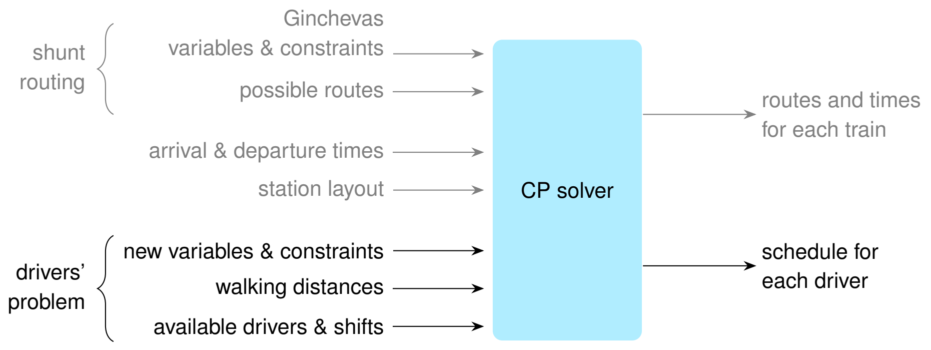
If the departure is in the next planning horizon, the 'departureYardIndex' is set to the yard where it should stay. Through this, the last position is not the departure track but simply the yard, and the train will not execute the last movement. To make this work, the last position is not set to the endpoint of the last route (as it could be that there is no last route), if the departure time is bigger than the end of the scenario.

Validation of the Minizinc model

The validation of Gincheva's Model was implemented in the HIP in C#, and the model in Minizinc did not have a validation. To validate the solution of Gincheva's Minizinc model and the expanded model (with splits and combines), the requirements made by Gincheva for the model in HIP are reused.

Improvement of code and running new stations

To make obtaining the instances of the stations easier, the pre-processing was automated. Additionally, for each new station, the HIP code to obtain the instances had to be adjusted.


shunt routing
Ginchevas variables & constraints
possible routes
arrival & departure times
station layout
drivers' problem
new variables & constraints
walking distances
available drivers & shifts
CP solver
routes and times for each train
schedule for each driver
Figure 3.6: Resulting combined model, composed of variables & constraints for the shunt routing problem as well as the drivers' problem.

3.2.4 Drivers' Problem

In addition to knowing which routes are used to get to the yards and back, a driver is needed to move the trains along the chosen routes. At NS, drivers either have a shift in which they drive only passenger trains or only shunt trains. Currently, the drivers' schedules for the shunt trains are computed in the local search of the HIP.

However, as the local search takes a long time to compute (around one hour for Amersfoort and more than 10 hours for Utrecht), there is an interest in computing some results for the drivers' schedules outside the local search, with a significantly lower computation time. This could be a benefit because it can happen that the train time schedules change two weeks before departure, for example, in the case of a football match in the country. Then, the shifts need to be changed, but the number of available drivers cannot be changed easily anymore. Therefore, it can be useful to combine the problems and be able to answer the question: 'Can I build a feasible solution with the number of shifts available?'

Definition drivers' problem. In the drivers' problem, schedules are computed for drivers to conduct trains to the yards and back. At the yards, the drivers may have to perform maintenance checks on the trains or split/combine train units. A **schedule** is a plan that denotes the time and sequence of each task that the driver has to complete in a shift. A **shift** is a set amount of time in which the driver works. A schedule is feasible if all the assigned tasks can be completed in the shift, taking into account not only the duration of the tasks but also the time to walk from one task to another.

The input consists of the number of available drivers and their shifts, and the walking distances across the station. The output is a schedule for each driver. The schedule will contain the trains to drive and their routes. The output is feasible if every driver's schedule is feasible and every train movement has an assigned driver at all times.

To implement the drivers' problem, new variables and constraints are added to Gincheva's SRP model. The resulting model, shown in Figure 3.6 solves the routing problem as well as the drivers' problem in the same step.

Assumptions

To compute a schedule for the drivers, the exact times and places of splits/combines and maintenance at the yard need to be known. However, exact yard movements and positions are only computed in the step after the yard selection and shunt routing. In order to still be able to compute the schedules, the following assumptions are made:

A1: Every driver is able to drive every train.

A2: Drivers always walk from one task to another, they are not allowed to ride a train as a passenger, for example, to reach the yard.

A3: The shifts do not contain breaks.

A4: The shifts do not include combining or splitting a train.

A5: The time of going from one track to another in the yard is the average time to move in between all tracks. This is the same for the platforms.

A6: Maintenance checks of the train will not be executed by the drivers.

3.2.5 Requirements

The model is complete when it satisfies these requirements:

R1: All the tasks inside a shift are possible to be executed within the time limits of that shift.

R2: The time planned for a driver to walk from A to B in a shift is not smaller than the walking distance defined by the input. The driver should always have enough time to walk from one location to the next.

R3: The shifts start at their start location and end at their end location. The walks to and from these areas are planned in the shifts.

R4: Every train move that is planned requires to be executed in exactly one shift. If a move is not executed, there should not be a shift assigned to it.

R5: In a shift, there are no gaps in which it is undefined what the driver is doing. A driver is either walking/waiting or driving.

R6: In a shift, no walks and drives overlap with each other. A driver cannot drive multiple trains simultaneously.

R7: Every shift contains only drives that a train also takes. (The driver should not be able to drive a train when that train movement does not exist in reality.)

These requirements are tested in a validation after running the model.

3.2.6 Formal Definition

Input

- S - set of shifts.
- $startshift_s$ - start time of shift $s \in S$.
- $endshift_s$ - end time of shift $s \in S$.
- $starttrack_s$ - start track of shift $s \in S$.
- $endtrack_s$ - end track of shift $s \in S$.

- *locations* - set of all yards at the node, the platforms, and staff areas.

  As the exact position in a yard is not relevant, but the position on the station (platform) is, this set contains all platforms individually, but only one element per yard.

- *numLoc* - number of yards, platforms, and staff areas.

- w_{yp} - walking time between y and p, where $y, p \in \{1, \dots, numLoc\}$.

- T - set of trains to be routed.

- $maxM_t$ - maximum number of movements for train $t \in T$. A movement of a train is equal to a reposition, and it always requires a driver.

Variables

The following variables are part of Gincheva's model for the SRP, and as they will also be used for the drivers' problem, they are listed again. For a full list of the variables and constraints of the old model, refer to [10].

- A^t_j - time interval variable denoting the times of the route that train t takes at movement j.

- e^t_j - variable denoting the location of train t after movement j. For $j = 0$, the starting location is described. Also called an endpoint.

- $s(), e(), d()$ - functions that return the start, the end, and the duration of interval variables, respectively.

All of the following variables are newly defined for the drivers' problem:

- $numD_s$ - the number of tasks in schedule $s \in S$, a task is the movement of the train that a driver has to execute.

There are two locations corresponding to the execution of a task in a schedule. This is because the driver has to walk to the starting point of the task, and therefore, the first repositioning of the driver is the walk to the train, and the second repositioning is the drive on the train.

- $pwalk^s_i$ - location of driver before walking to the task i, $i \in 0, \dots, numD_s$.
  For $i \in 1, \dots, numD_s$, it also denotes the arrival location of the previous task.

- $walk^s_i$ - time interval variable denoting the walking time to the task i or to the end location of the shift, and the waiting time before the next task, $i \in 0, \dots, numD_s$.
  The waiting time is added in this variable because it is allowed for the driver to have some waiting time in case the train cannot depart yet.

- $pdrive^s_i$ - location of driver after walking and before executing task i, i.e. driving a train to its destination, $i \in 0, \dots, numD_s$.
  For $i = numD_s$, it denotes the location at the end of the shift.

- $drive^s_i$ - time interval variable denoting the time of driving a train to its destination, executing task i, $i \in 0, \dots, numD_s - 1$.
  At the end of the shift, the driver still has to walk back to the staff area, there is one more 'walk' than 'drive'. Therefore, the range of i is smaller by one.

To assign the movement of a train to a schedule, notation from Gincheva's model is used, identifying the drive by the train and its movement.

- $chosenshift_j^t$ - variable denoting the shift in which the jth movement of train t is being executed, $chosenshift_j^t \in S$
- $chosenposition_j^t$ - variable denoting the position of move j of train t in the chosen shift, $chosenposition_j^t \in 0, \ldots, maxM * |T|$

Constraints

Constraints for start and end location of a shift
The first location should be the start location of the shift, before walking to the first task.

$$pwalk_0^s = starttrack_s \quad \forall s \in S \tag{3.1}$$

The last location of the driver should correspond to the ending location of their shift.

$$pdrive_{numD_s}^s = endtrack_s \quad \forall s \in S \tag{3.2}$$

Constraints for start and end times of a shift
The start time of the first walk should be the start time of the shift.

$$s(walk_0^s) = startshift_s \quad \forall s \in S \tag{3.3}$$

The end time of the last walk should be the end time of the shift.

$$e(walk_{numD_s}^s) = endshift_s \quad \forall s \in S \tag{3.4}$$

Constraints for the location of the driver before and after a drive
The location of the driver before a drive should be equal to the location of the corresponding train before it is moved. For all $t \in T$ and $j \in 0, \ldots, maxM_t$:

$$pdrive_i^s = e_j^t \text{ with } s = chosenshift_j^t \text{ and } i = chosenposition_j^t \tag{3.5}$$

After the drive (one could also say before a walk), the driver should be in the same location of the train after the movement. As an endpoint always denotes the location of a train before a move, the endpoint of the next move of the train must be chosen. For all $t \in T$ and $j \in 1, \ldots, maxM_t$:

$$pwalk_i^s = e_{j+1}^t \text{ with } s = chosenshift_j^t \text{ and } i = chosenposition_j^t + 1 \tag{3.6}$$

Constraint for duration and time of drive
The duration and time of a drive should be equal to the duration and time of the chosen route. For all $t \in T$ and $j \in 0, \ldots, maxM_t$:

$$drive_i^s = A_j^t \text{ with } s = chosenshift_j^t \text{ and } i = chosenposition_j^t \tag{3.7}$$

Constraints for walking and waiting time
The start of the walking time should be the end time of the previous drive. For the first walk, it is defined above as the shift starting time.
For all $s \in S$ and $i \in 1,\dots,numD_s$:

$$s(walk_i^s) = e(drive_{i-1}^s) \tag{3.8}$$

Similarly, the end time of a walk should be the start time of a drive. For all $s \in S$ and $i \in 0,\dots,numD_s - 1$:

$$e(walk_i^s) = s(drive_i^s) \tag{3.9}$$

The duration of the walk and eventual waiting time has to be at least the walking time required to go from one location to the next. For all $s \in S$ and $i \in 0,\dots,numD_s$:

$$d(walk_i^s) >= w_{yp} \text{ where } y = pwalk_i^s \text{ and } p = pdrive_i^s \tag{3.10}$$

Constraints for occurrence of drives for a schedule
If there is no drive to execute at a certain position in the shift, then the drive variable will be absent, and the duration of the drive will be 0. If there is a drive to execute, then the drive variable is present.
For all $s \in S$ and $i \in 0,\dots,numD_s - 1$:

$$absent(drive_i^s) \wedge d(drive_i^s) = 0 \qquad \text{for } i \geq numD_s \tag{3.11}$$
$$occurs(drive_i^s) \qquad \text{for } i \leq numD_s \tag{3.12}$$

Constraint to assign a shift to a move
If the move of a train is present, then there has to be a shift in which the move is executed.
For all $t \in T$ and $m \in 0,\dots,maxM_t$:

$$occurs(A_m^t) \implies chosenshift_m^t \neq -1 \tag{3.13}$$

Constraint for the number of tasks in a schedule
This constraint contains the global constraint `count_eq(array [X] of var int: x, var int: y, var int: c)`. This constraint takes an array of variables and two integer variables. It constrains the array to have exactly c occurrences of y in X. As $numD_s$ is the number of drives in a shift, among all the variables denoting the chosen shift for a train move, shift s must occur exactly $numD_s$ times.
For all $s \in S$:

$$count\_eq(\left[chosenshift_m^t | t \in T, m \in \{0,\dots,maxM_t\}\right], s, numD_s) \tag{3.14}$$

3.2.7 Alternative Modelling of Drivers' Problem

In the following, an alternative way of modelling the drivers' problem is presented. Instead of having only the train variables reference the shift they are linked to, the linking is done twice, from the train variables to the shift variables and back. This could raise the models' complexity, and the experiments section investigates whether this implementation yields a higher runtime. All changes to the model are mentioned here. If there is no mention of a certain constraint, it means that it is not changed.

Changes to Variables

As the shift variables are now linked to the moves of the trains, the variables *tchosendrive* and *mchosendrive* are added. Each move of a train still contains the *chosenshift* variable, denoting the shift in which the drive is executed (as before), and the *chosenposition* variable is removed.

- $tchosendrive_i^s$ - variable denoting the train that is driven for task i, $i \in 0,\dots,numD_s - 1$, $tchosendrive_i^s \in T$
- $mchosendrive_i^s$ - variable denoting the movement of the train that is driven for task i, $i \in 0,\dots,numD_s - 1$, $mchosendrive_i^s \in maxM_t$

Changes to Constraints

First, constraint 3.5 is changed. The constraint is repeated here:
For all $t \in T$ and $j \in 0,\dots,maxM_t$:

$$pdrive_i^s = e_j^t \text{ with } s = chosenshift_j^t \text{ and } i = chosenposition_j^t$$

The new constraint also sets the location of the driver before a move, but now it involves the new variables:
For all $s \in S$ and $i \in 0,\dots,numD_s - 1$:

$$pdrive_i^s = e_j^t \text{ with } t = tchosendrive_i^s \text{ and } j = mchosendrive_i^s \tag{3.15}$$

Next, the constraint setting the location of the driver after a move, 3.6, is changed from:
For all $t \in T$ and $j \in 1,\dots,maxM_t$:

$$pwalk_i^s = e_{j+1}^t \text{ with } s = chosenshift_j^t \text{ and } i = chosenposition_j^t + 1$$

to the following, adding the *tchosendrive* and *mchosendrive* variables:
For all $s \in S$ and $i \in 1,\dots,numD_s$:

$$pwalk_i^s = e_j^t \text{ with } t = tchosendrive_{i-1}^s \text{ and } j = mchosendrive_{i-1}^s + 1 \tag{3.16}$$

After the locations of the drivers are set, the duration and times for the moves of the trains have to be set. In the previous model, it is described with the following constraint 3.7.
For all $t \in T$ and $j \in 0,\dots,maxM_t$:

$$drive_i^s = A_j^t \text{ with } s = chosenshift_j^t \text{ and } i = chosenposition_j^t$$

In the new model, the constraint is adjusted to be represented by the new variables and without the removed ones:
For all $s \in S$ and $i \in 0,\dots,numD_s - 1$:

$$drive_i^s = A_j^t \text{ with } t = tchosendrive_i^s \text{ and } j = mchosendrive_i^s \tag{3.17}$$

A constraint responsible for the 'double linking' of the train and shift variables, is added:
For all $s \in S$ and $i \in 0,\dots,numD_s - 1$:

$$chosenshift_j^t = s \text{ with } t = tchosendrive_i^s \text{ and } j = mchosendrive_i^s \tag{3.18}$$

Lastly, constraints 3.11 are changed. The old constraints are repeated in the following.
For all $s \in S$ and $i \in 0, \ldots, numD_s - 1$:

$$\begin{aligned} absent(drive_i^s) \wedge d(drive_i^s) = 0 \qquad & \text{for } i \geq numD_s \\ occurs(drive_i^s) \qquad & \text{for } i \leq numD_s \end{aligned}$$

The new constraint adds setting the train for a task in a drive:
For all $s \in S$ and $i \in 0, \ldots, numD_s - 1$:

$$absent(drive_i^s) \wedge d(drive_i^s) = 0 \wedge tchosendrive_i^s = -1 \qquad \text{for } i \geq numD_s \tag{3.19}$$

$$occurs(drive_i^s) \wedge tchosendrive_i^s \neq -1 \qquad \text{for } i \leq numD_s \tag{3.20}$$

3.3 Hypotheses

The creation of the combined model with the research questions leads to the following hypotheses:

3.3.1 The model computes the solution to the drivers' problem and shunt routing correctly.

The model should compute a solution that is in accordance with the defined requirements.

3.3.2 The addition of the drivers' problem to the SRP will result in a higher runtime for the model.

As the created model will solve two problems that are intertwined with each other, the increased complexity will lead to a higher runtime.

3.3.3 The performance of the model changes after connecting the train and driver variables bidirectionally instead of unidirectionally.

In the base model, the moves are connected to the shifts, but the shifts are not connected to the moves. This connection could also be bidirectional. In a modified implementation of the model, the moves of the trains are linked to the drivers' shifts in two ways: Firstly, each move links to a shift in which it gets executed, and secondly, each task in a shift references the train and its move, which is executed by the shift. This double linking could be causing a higher complexity, and increase the runtime subsequently.

3.3.4 The fixed section occupations have an influence on the runtime of the model.

The fixed section occupations describe which sections are occupied by fixed through trains. When deciding the routes of the plannable trains, these fixed section occupations have to be taken into account. A large amount of section occupations could lead to a higher runtime, as there are more limitations to feasible solutions.

3.3.5 Using variable and value selection strategies for the waiting time of the drivers in between tasks will reduce the runtime.

If a driver has a high waiting time before executing the next task, the driver can execute fewer tasks overall, raising the probability of exploring an infeasible solution. If the solver starts with a lower value for the waiting time, a feasible solution could be found quicker, as solutions with more tasks in a shift are explored first.

3.3.6 Using variable and value selection strategies for the number of tasks in a shift will reduce the runtime.

The domain of the variable denoting the number of tasks in a shift ranges from 0 to the maximum number of moves of all trains. As it is highly unlikely that a single driver will execute all these in one shift, a search strategy could prevent from searching these infeasible solutions. In these search strategies, only the smaller or median values will be assigned, and this could lead to finding a feasible solution faster.

3.3.7 Using variable and value selection strategies for the chosenshift variable will reduce the runtime.

The chosenshift variable is assigned every time a move is executed, and it describes the corresponding shift. With selection strategies, the solver can be guided for example to first 'fill up' the first shift with tasks and so on. This could avoid assigning tasks to schedules that result in an infeasible solution and therefore improve performance.

3.3.8 Using variable and value selection strategies for the parking on the departure platform will reduce the runtime.

Spending more time on the departure platform means that in the meantime, no other train can access the platform. This could lead to infeasible solutions. Assigning smaller values first to the time spent on the departure platform could therefore improve the performance. This hypothesis was put forward by Gincheva, and is now tested in the combined model.

Chapter 4

Experiments and Results

The following experiments aim to analyze the performance of the model that combines the drivers' problem and the SRP under different conditions. Four stations are available for the experiments: Vlissingen, Enkhuizen, Enschede, and Amersfoort. First, the model is run without any modifications on all instances in order to examine the difference between the individual instances and the stations 4.2. Then, the model is run with multiple solvers, comparing their performance 4.3. Overall, CP-SAT with free search and multiple threads performs the best. The influence of different seeds is explored in Section 4.4, which is shown to be rather insignificant. In the next Section 4.5, the drivers' problem combined with the SRP is compared to the SRP alone, drawing conclusions and comparisons to Gincheva's model. It seems that the addition of the drivers' problem to the SRP raises the complexity significantly and subsequently the runtime. Different ways of linking the driver and train variables in the model are examined in Section 4.6, resulting in the conclusion that a unidirectional linking of the variables yields a significantly lower runtime than a bidirectional linking. During the examination of the instances, a correlation is observed between the runtime and the number of fixed section occupations, but a connection is disproved in further experiments 4.7 where the instances are run without any fixed section occupations. With the use of variable and value selection strategies, the solver can be guided to obtain a solution faster. Strategies on four different variables of the model are tested: The walk and wait variable 4.8.1, the numD variable 4.8.2, the chosenshift variable 4.8.3, and the variable for the last stay on the departing platform 4.8.4. Using a strategy for the walk and wait variables yields an overall improved performance for the Enkhuizen and Enschede stations, however, the Vlissingen station results in higher runtimes. When using the right type of strategies on the numD variable, the runtime never worsens on all instances and improves performance on some instances. The strategies on the chosenshift variable yield ambiguous results, improving the runtime in some instances but worsening it in others. The stay on the departing platform with different strategies was also examined by Gincheva, and her results - stating that the strategies only work well for Enkhuizen - could be reproduced.

The experiments examine the relationship to Gincheva's model, analyze the model's properties and their impact on the runtime, and demonstrate that performance can be further improved through certain search strategies.

In every experiment, each instance is run five times, and in the plots, the variability of the results can be seen by the error bar with a 95% confidence interval. The runs are all executed on DelftBlue [7], with 1 CPU and 32 GB of RAM. If a solver is used with multiple threads, 4 CPUs are used. The runs time out after 15 minutes (900 seconds). If there is a time-out, the runtime is set to 900 seconds in the plot. In all experiments, the instances are sorted by the number of trains. If there are two instances with the same number of trains, the date is used as the order. In the instances, there is only one route for each move, so the solver does not need to choose between different routes. Furthermore, this route is the fastest one and could contain a reversal. The experiments are run in Minizinc with version 2.9.0, and the solver CP-SAT with version 9.11.4210 and Chuffed with version 0.13.2. For each run in the experiments that yields a satisfiable solution, the result gets validated according to the requirements in 3.2.5. Table 4.1 presents an overview of all experiments.

Experiment type	Experiment	Solver	Stations
Analysis of base model	Comparison of instances	cp-sat, free search	Vlissingen, Enkhuizen, Enschede, Amersfoort
	Comparing solvers	chuffed, cp-sat, multi threaded, free search	Vlissingen, Enkhuizen, Enschede, Amersfoort
	Comparison to Gincheva	cp-sat, free search	Vlissingen, Enkhuizen, Enschede, Amersfoort
	Different seeds	cp-sat, free search	Vlissingen, Enkhuizen, Enschede, Amersfoort
Modification of model	Bidirectional linking	cp-sat, free search	Vlissingen, Enkhuizen, Enschede, Amersfoort
	Removal of section occupations	cp-sat, free search	Vlissingen, Enkhuizen, Enschede, Amersfoort
Variable and value selection strategies	Waiting and walk variable	cp-sat, free search	Vlissingen, Enkhuizen, Enschede
	Number of tasks in shift	cp-sat, free search	Vlissingen, Enkhuizen, Enschede
	Chosen Shift variable	cp-sat, free search	Vlissingen, Enkhuizen, Enschede
	Parking on departure platform	cp-sat, free search	Vlissingen, Enkhuizen, Enschede

Table 4.1: Experiments Overview

4.1 Instances

The instances used are real instances, modified with the preprocessing necessary to run the underlying CP model for the SRP, mentioned in Section 3.2.2. This preprocessing removes all plannable trains that depart or arrive on a platform where a through train is parked. Furthermore, some trains in new instances resulted in an infeasibility for the SRP. Therefore, these trains are removed as well. In the experiments, when the 'smallest' or 'largest' instance of a station is mentioned, it refers to the number of trains and therefore the leftmost or rightmost instance in the plot and table. The following stations are used in the experiments: Vlissingen, Enkhuizen, Enschede, and Amersfoort. An instance is a timeframe (mostly a day), and contains all the trains that have to be planned and all trains that are fixed during that time, and the available shifts.

In order to obtain more metrics to analyze the results, some attributes of each instance are detailed in a table. Table 4.2 explains each attribute.

The maximum number of movements is dependent not only on the train type, but also on where the arrival and departure track is. A base shunt train normally has a maximum of two movements, one from the platform to the yard and one back. However, it could also be that the train 'arrives' or 'departs' at the yard. Then, there is only one maximum movement to be made. Therefore, instances with the same amount of trains can have a different number of moves that can be made. This leads to more assignments to a shift and possibly to a higher runtime. The average number of the maximum number of movements per train allows for some analysis of the movements each train has to make.

	Date of the instance
Timeframe	Duration of the start to the end of the scenario
Trains	Number of trains in the instance. This number includes every train type. For example, if a train gets split into two parts and these two parts depart, it will get counted as three trains (One arrival train and two split trains).
Base Shunt Trains	Number of trains that do not get split and/or combined.
Arrival Trains	Number of trains that are going to be split.
Split Trains	Number of trains that have been split.
Combined Trains	Number of trains that are going to get combined.
Departure Trains	Number of trains that have been combined.
Split Combined Trains	Number of trains that are split and combined.
Trains combined to fixed through train	Number of trains that are combined to a fixed train at the platform. The combining does not have to be planned, but the model must take into account from which side the train must reach the platform.
Trains split from fixed through train	Number of trains that are split at the platform from a fixed train. The splitting is not taken into account in the model, but the side and time the plannable train can leave the platform is relevant.
Shifts	Number of shifts in the instance. One shift gets executed by one driver.
Locations	Number of platforms, tracks, staff areas and yards that a driver can be at.
Yards	Number of yards.
Section occupations	Number of intervals that describe occupied sections. When a section is occupied, no other train can access it during that interval.
Max number of sections in route	The maximum amount of sections one route of a train can have.
Average max number of movements per train	This is the average number of maximum movements that a train can possibly make.
Total max number of movements	The maximum amount of moves that can be planned in the instance.

Table 4.2: Explanation of each analyzed attribute

4.1.1 Vlissingen

	27/05	25/05	29/05	17/05	23/05
Timeframe	24h	24h	24h	24h	24h
Trains	9	10	11	13	24
Base Shunt Trains	9	10	8	13	24
Arrival Trains	0	0	0	0	0
Split Trains	0	0	0	0	0
Combined Trains	0	0	2	0	0
Departure Trains	0	0	1	0	0
Split Combined Trains	0	0	0	0	0
Trains combined to fixed through train	0	0	0	0	0
Trains split from fixed through train	0	0	0	0	0
Shifts	5	3	5	3	5
Locations	6	7	6	6	6
Yards	1	1	1	1	1
Section occupations	1003	732	744	673	964
Max num sections of route	7	7	7	7	7
Average max number of movements	1.889	1.5	1.455	1.769	1.917
Total max number of movements	17	15	16	23	46

Table 4.3: Attributes for all Vlissingen instances

4.1.2 Enkhuizen

	29/05	14/05	27/05	24/04	26/05
Timeframe	24h	24h	24h	24h	33.5h
Trains	13	14	14	15	16
Base Shunt Trains	13	11	14	12	16
Arrival Trains	0	1	0	1	0
Split Trains	0	2	0	2	0
Combined Trains	0	0	0	0	0
Departure Trains	0	0	0	0	0
Split Combined Trains	0	0	0	0	0
Trains combined to fixed through train	3	6	5	6	9
Trains split from fixed through train	0	2	6	4	7
Shifts	5	4	5	5	6
Locations	5	5	4	5	4
Yards	1	1	1	1	1
Section occupations	488	514	499	509	730
Max num sections of route	6	5	5	6	5
Average max number of movements	1.615	1.5	1.857	1.733	1.875
Total max number of movements	21	21	26	26	30

Table 4.4: Attributes for all Enkhuizen instances

4.1.3 Enschede

	06/05	23/04	14/05
Timeframe	10h46m	24h	24h
Trains	28	30	30
Base Shunt Trains	4	9	7
Arrival Trains	1	0	2
Split Trains	2	0	0
Combined Trains	14	15	11
Departure Trains	7	6	6
Split Combined Trains	0	0	4
Trains combined to fixed through train	0	3	3
Trains split from fixed through train	3	11	9
Shifts	9	12	12
Locations	11	15	12
Yards	1	1	1
Section occupations	632	2271	1920
Max num sections of route	13	13	13
Average max number of movements	0.821	1.167	1
Total max number of movements	23	35	30

Table 4.5: Attributes for all Enschede instances

4.1.4 Amersfoort

	01/11
Timeframe	24h
Trains	26
Base Shunt Trains	9
Arrival Trains	2
Split Trains	1
Combined Trains	6
Departure Trains	5
Split Combined Trains	3
Trains combined to fixed through train	0
Trains split from fixed through train	0
Shifts	15
Locations	21
Yards	1
Section occupations	16148
Max num sections of route	20
Average max number of movements	1.23
Total max number of movements	32

Table 4.6: Attributes for the Amersfoort instance

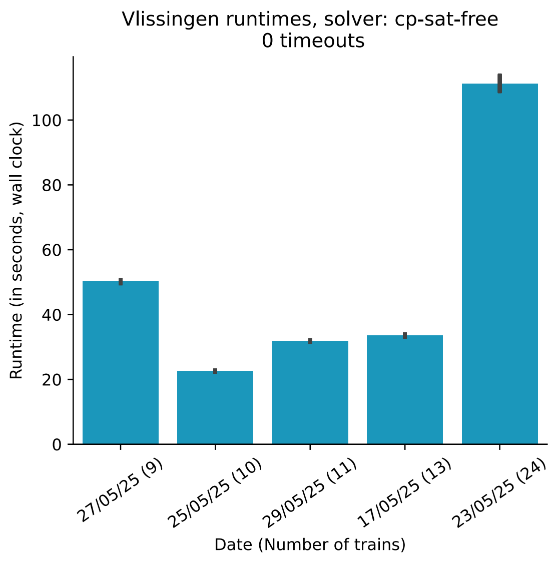
4.2 Comparing Instances

In these experiments, all instances are run with the solver CP-SAT with free search, a single thread, and five different seeds, each seed five times. The goal is to compare the instances and analyze them while looking at their attributes. As each instance has different attributes, it can be examined whether certain station or instance characteristics influence the runtime. The y-axis describes the runtime of each instance, which is depicted on the x-axis.

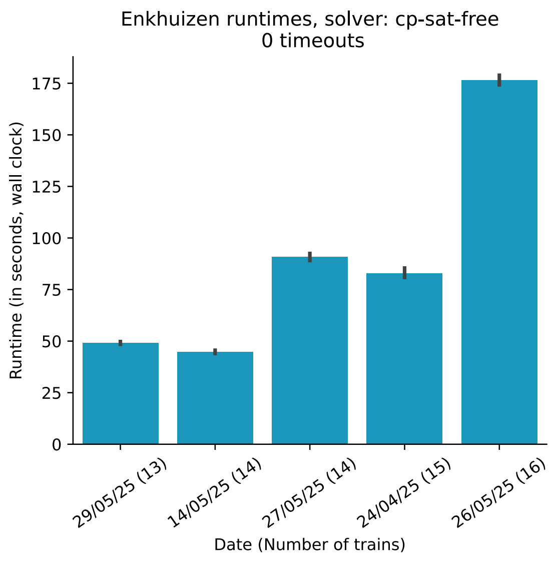
The runtimes for the Vlissingen station are depicted in Figure 4.1a. The instance that has a significantly higher amount of trains (24 vs 9-13 trains) also has a significantly higher runtime. However, the number of trains does not seem to be the only attribute leading to a high runtime: the second-highest runtime belongs to the smallest instance with 9 trains. With almost 50 seconds, the instance with 9 trains is twice as slow as the instance with 10 trains. The reason for this could be the number of section occupations. While the instance with 10 trains has 732 section occupations, the instance with 9 trains has 1003. Another factor contributing to the bad performance of the smallest instance could be the maximum number of total moves. In the 9 trains instance, there are a maximum of 17 moves, and in the 10 and 11 trains instances, there are a maximum of 15 and 16 moves, respectively. To recall: these numbers are different because, depending on its type, a train can have a varying number of possible moves. So even though the number of trains is smaller, there are more moves that can be executed and subsequently have to be assigned to a shift. This could also explain the high runtime for the largest instance: even though it has fewer fixed section occupations than the smallest instance, the maximum number of moves is almost three times as big, at 46.

The Enkhuizen instances, depicted in Figure 4.1b, have less variation in the number of trains, ranging from 13 to 16. Two instances contain 14 trains, and as they have a significant difference in performance (45 seconds for 14/05 vs 90 seconds for 27/05), it shows that the number of trains alone is not informative about the runtime. The two instances do not differ much in the number of section occupations, however, the 14/05 instance has 21 maximum movements, while the 27/05 instance has 26. This could show that the number of movements has an influence on the performance. This would make sense, as every movement needs to be assigned to a multitude of variables: a shift, a position in this shift, and start and end times that do not coincide with other section occupations. The largest instance, 26/05 with 16 trains, has by far the highest runtime with around 175 seconds. With 16 trains, there is only 1 train difference from the instance with 15 trains, however, the remaining attributes do show a higher variety in the two instances: While the instance with 15 trains has 509 fixed section occupations, the 16 trains instance has 730. Additionally, the number of maximum movements is higher in the bigger instance with 30 movements, and 26 movements in the smaller instance with 15 trains. Another contributing factor to the high runtime of the largest instance in comparison to the remaining instances could be the timeframe of the instance. While it is 24 hours for the other instances, the largest instance spans over a time of 33.5 hours. Lastly, this slowest instance contains only trains that are split from or combined to a fixed through train at the platform. This also adds complexity as it restricts the train to only enter/leave the platform from one side and/or when the through train is not yet departed/arrived.

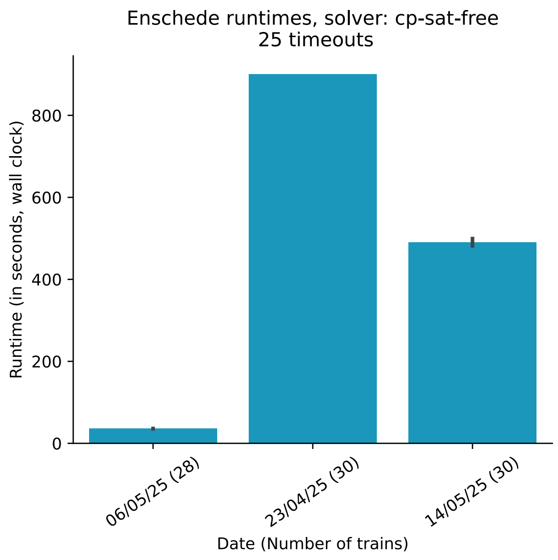
The Enschede station has three instances with 28, 30, and 30 trains. However, as seen in Figure 4.1c, the 28 trains instance has a runtime of around 30 seconds, while the 23/04 instance always times out, and the 14/05 instance has a runtime of around 500 seconds. The attributes explain the large difference in performance, and show again that the number of trains alone allows little speculation about the runtime. The maximum movements of the small instance are 23, compared to 35 and 30 in the bigger instances. The most substantial difference however, lies in the section occupations: While the small instance contains 632, the two bigger instances have 2271 and 1920 section occupations. The timeframe of the instances also could influence the performance: The smaller instance only has a timeframe of 10 hours and 46 minutes, while the other two instances span over 24 hours. This could increase the search space and subsequently lead to a higher runtime.



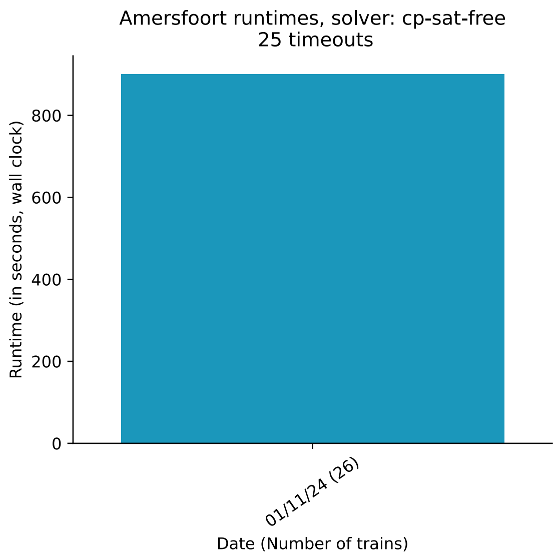
(a) Vlissingen



(b) Enkhuizen



(c) Enschede



(d) Amersfoort

Figure 4.1: Runtimes for all stations with the CP-SAT solver and free search

Unfortunately, the instance for Amersfoort depicted in Figure 4.1d always times out. Looking at the attributes of the instance, however, reveals some possible reasons for this high runtime: There are 16148 section occupations, almost 8 times as many as the bigger Enschede instances (23/04 and 14/05). If the section occupations have an influence on the runtime, this could be the reason for the timeouts. Additionally, to the sections occupied by fixed trains, one route of a train can have up to 20 sections that it needs to occupy. For Enschede, this number is 13, and for Enkhuizen and Vlissingen, it is around 5-7. This means that when deciding on a route, more sections and their availability need to be taken into account, possibly raising the complexity.

Overall, it seems that there are multiple attributes that lead to a higher runtime: The results could suggest that the number of fixed section occupations influences the performance, as stated in Hypothesis 3.3.4. This is tested in Section 4.7. The number of trains is not informative, this could be because the number of overall movements differs depending on the types of trains, and not the number of trains.

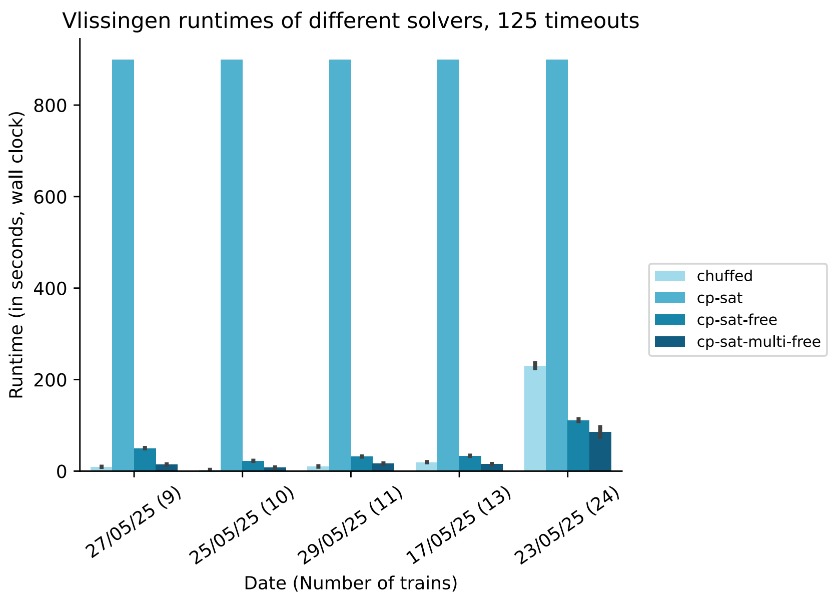


Figure 4.2: Comparison of Different Solvers for Vlissingen

Subsequently, the maximum number of moves seems to have an influence on the performance, stronger than the number of trains. The number of trains that are split from or combined to a fixed through train could add complexity in the model, as well as the specific types of trains in an instance. However, there is not enough variety in the instances to be able to be more specific. The number of shifts does not seem to have a significant influence. This could be because of the pre-processing. As the pre-processing removes some trains, it could be that instances that have a higher number of shifts also previously had a higher number of trains or moves. Lastly, the size of the station could also have a contribution to the performance: Amersfoort is the largest station, which is also shown by the fact that a route can consist of up to 20 section occupations. This leads to a higher complexity when planning the routes, as more sections could be occupied throughout.

4.3 Comparing Different Solvers

In this experiment, the performance of different solvers is compared. Each instance is run with five different seeds, five times. The solvers tested are: Chuffed (shown in plot as chuffed) and OR-Tools with three different settings (shown in plot as cp-sat, cp-sat-free, and cp-sat-multi-free). Other solvers, namely Coinbc, Gecode, and Highs have been tested as well, but are not included in the experiments due to weak performance.

Chuffed is a CP solver based on lazy clause generation. In lazy clause generation, clauses get generated upon discovering a failure, in order to avoid making the same decision again. This allows for a high-level model but leverages the advantages of SAT solvers (using clauses) at the same time.

The CP-SAT solver by Google OR-Tools uses lazy clause generation as well, combining constraint programming with SAT-solving techniques. CP-SAT can be instructed to do a free search, which enables a strategy that can be more efficient. In this strategy, multiple subsolvers are used at the same time. Moreover, the CP-SAT solver is also run with multiple threads enabled (cp-sat-multi-free). In this case, 4 threads are used.

In Figure 4.2, the solvers are compared on all instances of the Vlissingen station. The Chuffed solver solves the four smaller instances in under 20 seconds, however, the largest instance has a runtime of ∼210 seconds. On the three smaller instances, Chuffed even outperforms all solvers. CP-SAT with no solver option times out on every instance. Enabling the free search, however, significantly decreases the runtime. Interestingly, the instances that have the highest runtime for CP-SAT-free are the smallest

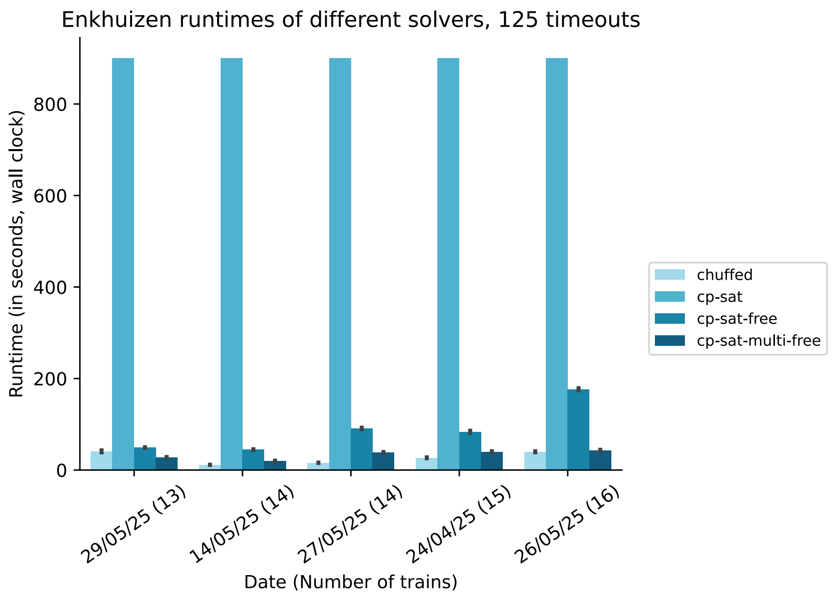


Figure 4.3: Comparison of Different Solvers for Enkhuizen

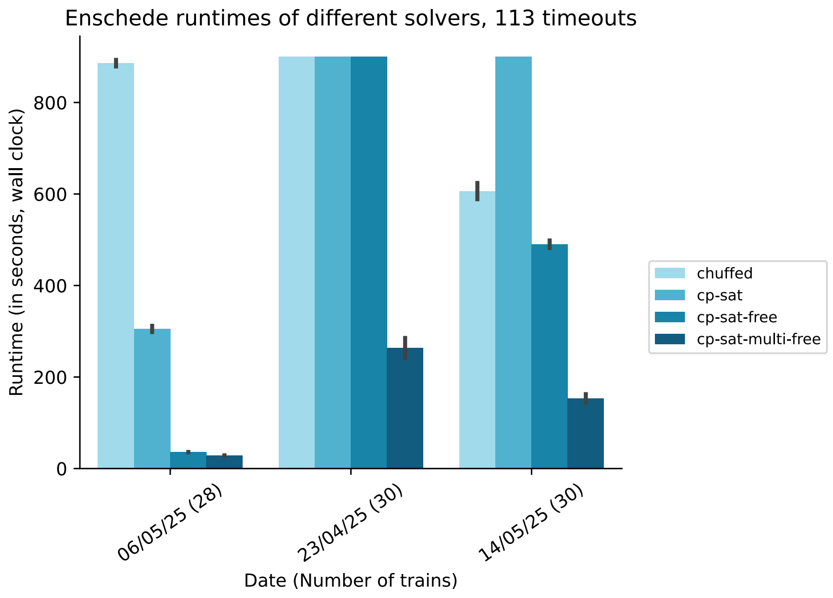


Figure 4.4: Comparison of Different Solvers for Enschede

instance with 9 trains and the largest instance with 24 trains. One possible explanation could be that these two instances have the highest number of section occupations, as can be seen in Table 4.1.1. The influence of the section occupations is analyzed in Section 4.7. Chuffed or CP-SAT with free search and multiple threads are overall the fastest solvers for the Vlissingen instances.

Figure 4.3 depicts the comparison of the different solvers for the Enkhuizen station. The Chuffed solver is able to outperform all other solvers for the four biggest instances. Just as for Vlissingen, CP-SAT with no options times out on every instance. Using free search with CP-SAT again results in a significant speed-up. Using multiple threads leads to a faster computation in comparison to using a single thread with CP-SAT-free; it at least halves the runtime, and for the largest instance (26/05), the model is almost 5 times faster. For the Enkhuizen instances, Chuffed and CP-SAT with free search and multiple threads perform the best.

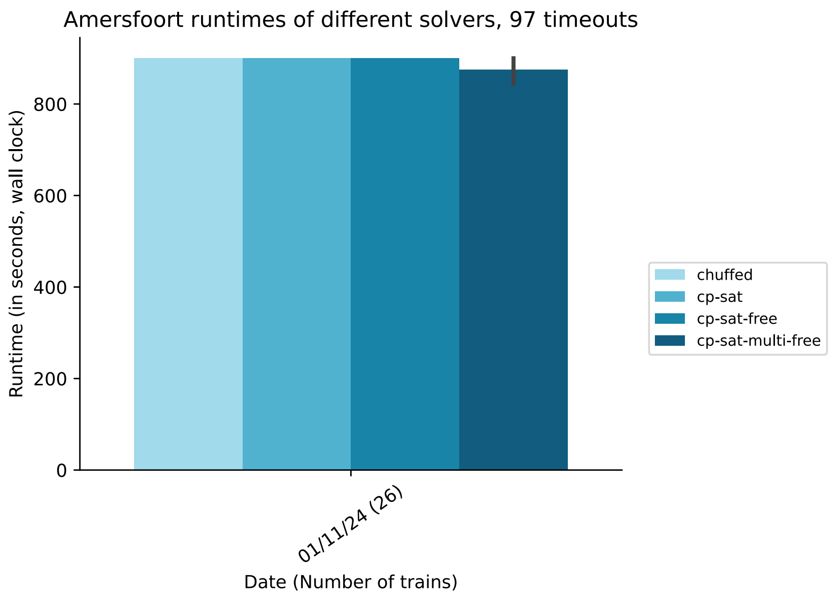


Figure 4.5: Comparison of Different Solvers for Amersfoort

The runtimes of the Enschede instances can be seen in Figure 4.4. The Chuffed solver overall performs worse than in the previous stations, by having by far the highest runtime for the smallest instance and timing out on the 23/04 instance. CP-SAT with no solver options is able to solve the smallest instance without a timeout, however still times out on the remaining instances. Adding free search to CP-SAT again significantly reduces the runtime, except for the 23/04, which yields a timeout and therefore is not known. CP-SAT with free search and multiple threads is again the fastest solver for all instances, solving every instance in under 300 seconds.

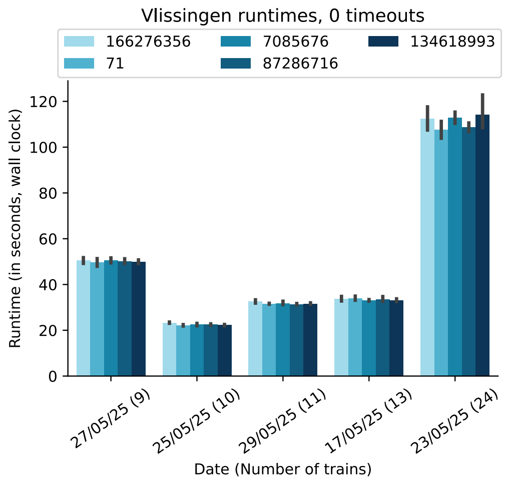
The Amersfoort instance, depicted in Figure 4.5, times out with Chuffed, CP-SAT with and without free search. For CP-SAT with free search and multiple threads, it times out on 22 out of 25 runs, yielding again the best performance out of all tested solvers.

Overall, Chuffed seems to work well on smaller instances (up to 16 trains). In some instances (Vlissingen 25/05, 27/05; Enkhuizen 14/05, 27/05), it achieves a better runtime than CP-SAT with free search and multiple threads, while only using one thread. As CP-SAT without free search and a single thread times out for every instance except one, it can be said that adding free search to the solver substantially reduces the runtime for the search. For smaller and medium instances, CP-SAT with a single thread and free search can return a solution without timing out, but it is consistently slower than when used with multiple threads. The solver CP-SAT with free search and multiple threads results in the lowest runtimes. However, since using multiple threads can result in a high variety of the results, and if the model were implemented in HIP, only one thread would be used, CP-SAT with free search, and a single thread is used for the experiments.

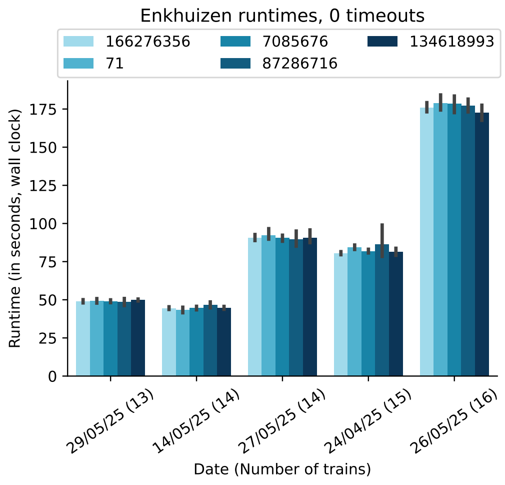
4.4 Comparing Different Seeds

The chosen seed can have an impact on the runtime. In these experiments, the instances are run with five different seeds, and the influence of the seed will be analyzed. The solver used is CP-SAT with free search and a single thread.

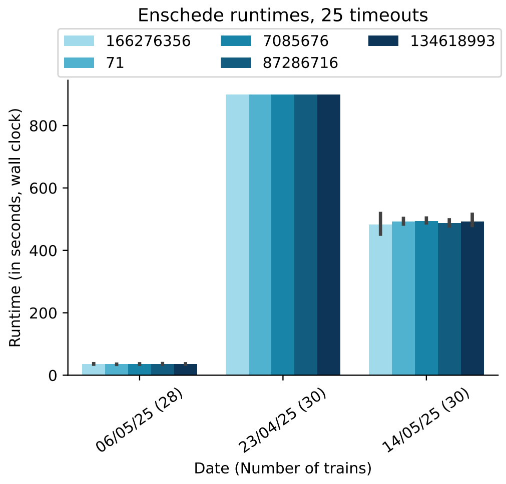
Figure 4.6a and 4.6b show the plots comparing the runtimes of different seeds for Vlissingen and Enkhuizen, respectively. Neither of the stations exhibits significant varying behaviour when executed with different seeds.



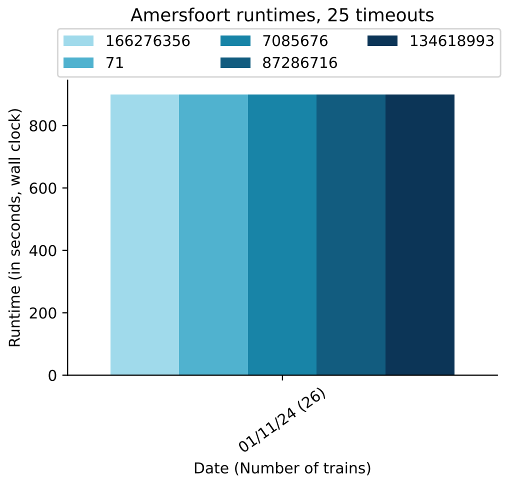
(a) Vlissingen



(b) Enkhuizen



(c) Enschede



(d) Amersfoort

Figure 4.6: Runtimes of the model with different seeds

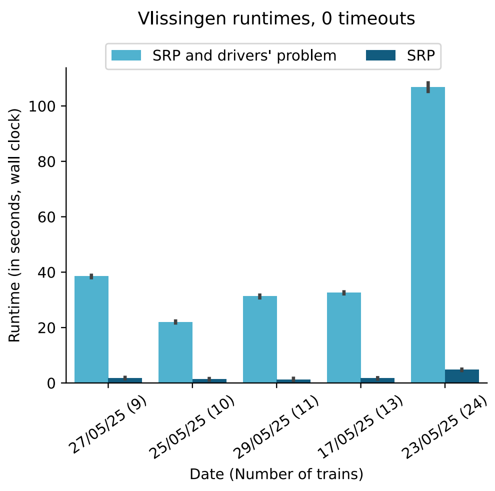
The model times out for every seed on the Amersfoort station, depicted in Figure 4.6d. For the Enschede station shown in Figure 4.6c, the seeds all result in a similar runtime for the smallest instance and the 14/05 instance, and the remaining instance times out with every seed.

Overall, it seems that the choice of seed does not influence the performance to a great extent.

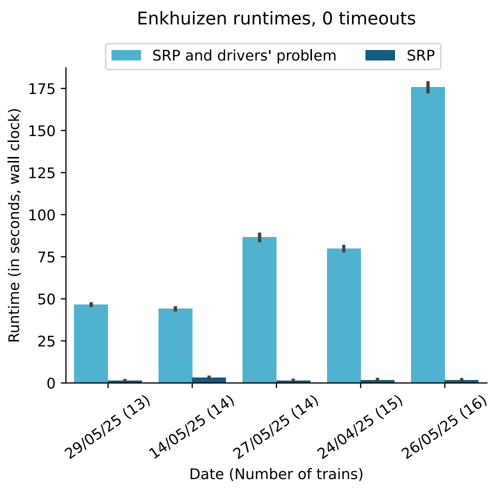
4.5 Model with and without the Drivers' Problem

In the following experiments, the model is run on all instances and stations as usual (SRP and drivers' problem combined) and with the removal of the constraints and variables for the drivers' problem in order to test Hypothesis 3.3.2. What remains is a model that solves the shunt routing problem. This model is mostly Gincheva's model, extended to work on all stations. All the experiments are run with five seeds, each seed five times. The solver used is CP-SAT with free search and a single thread.

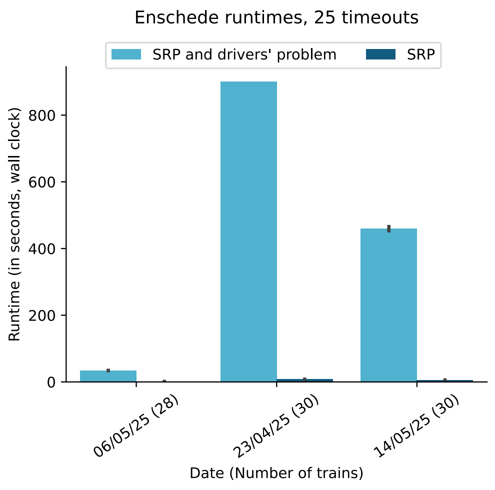
What comes to notice directly: In all instances, the runtimes differ significantly when the drivers' problem is removed. In Figure 4.7a for Vlissingen, the runtimes decline from 20-40 seconds to around 1



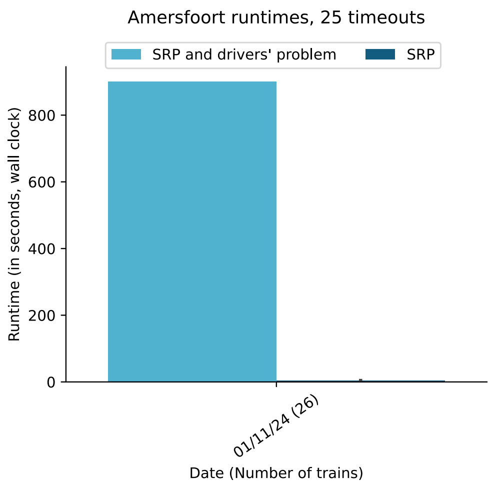
(a) Vlissingen



(b) Enkhuizen



(c) Enschede



(d) Amersfoort

Figure 4.7: Runtimes of the model with and without the drivers' problem

second for the four smaller instances. The largest instance (23/05) has a runtime of 4 seconds, compared to around 110 seconds with the drivers' problem.

In the Enkhuizen station 4.7b, the runtimes with the drivers' problem are around 40-175 seconds for all instances, and decrease to around 1-3 seconds without the drivers' problem.

Figure 4.7c depicts the Enschede station. The smaller instance (06/05) that gets solved in $\sim$34 seconds with the drivers' problem decreases to a runtime of $\sim$1 second. Even the instance that times out normally (23/04) gets solved in about 7 seconds. The 14/05 instance obtains a speed up of more than a factor of 90, from $\sim$460 seconds to around 5 seconds.

In Figure 4.7d, the instance for Amersfoort is shown. With the drivers' problem, the time runs out, but without it, the runtime is around 3.4 seconds.

The high magnitude of difference in the runtimes could be attributed to multiple reasons. One reason could be that the drivers' problem has a very large search space compared to the SRP. Next to deciding the movements and times in the SRP, the drivers also have to walk to their task, adding complexity and variables. Furthermore, the tasks need to be assigned to the shifts, creating many possible assignments of tasks to shifts. Another reason could be the nature of combining two problems. In the combination of the SRP with the drivers' problem, there are two different aspects that have to be scheduled: the trains and the drivers. When the trains alone have to be scheduled, the only thing to consider is the other trains. When the drivers' schedules are added, for every possible train schedule, a fitting driver schedule has to be found and vice versa. This combination of the two problems could lead to a high complexity.

Comparison to Gincheva's results

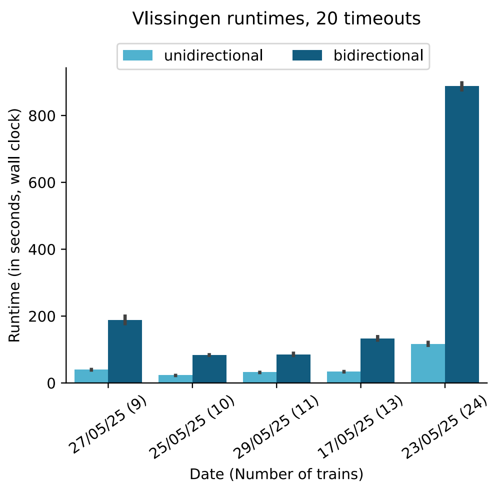
Gincheva tested the model for the SRP on one Amersfoort and one Enkhuizen instance. The Amersfoort instance had 23 trains after the preprocessing and reached a runtime of around 242 milliseconds in the HIP. It was not run in Minizinc. When reproducing this, our model for the SRP resulted in a runtime of around 3.4 seconds in Minizinc for the Amersfoort instance of 26 trains. This higher runtime cannot be attributed only to the higher number of trains. One factor could be the environment of the model. When running the Enkhuizen instance in the OR-Tools API (in HIP) and in Minizinc, Gincheva observed a significantly higher runtime, from around 450 seconds in HIP to timing out after 900 seconds in Minizinc. Therefore, our observed higher runtime could be attributed to the usage of Minizinc instead of C# with the OR-Tools API. Overall, Gincheva's results were reproduced successfully with similar results in these experiments. The Enkhuizen instance for Gincheva's model had 12 trains and a runtime of 443 seconds in the HIP, and in Minizinc with CP-SAT with no solving options, it timed out. Running our adapted model for the SRP in Minizinc with free search yields a runtime of around 1-2 seconds for all instances. This speed-up can be attributed partially to the use of free search, which has proven in previous experiments to improve the runtime. The other reason, and probably the most significant, is that in our instances, the three yards are seen as one, while Gincheva's instance contains three yards. Experiments conducted by Gincheva suggested that the number of yards greatly increases the runtime. Therefore, our model resulted in a lower runtime. To further explore this hypothesis, future experiments could be run by comparing the same instance with three yards and three yards combined into one.

4.6 Unidirectional vs. Bidirectional Linking

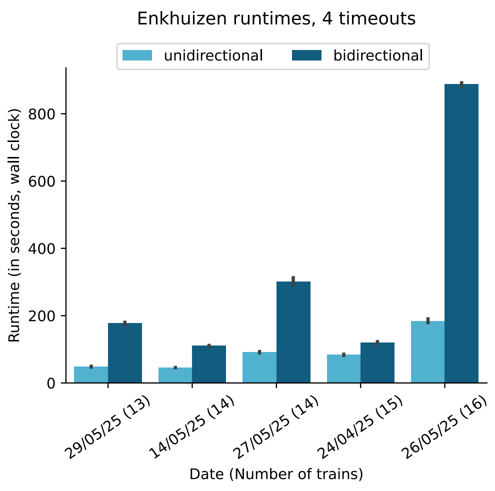
In these experiments, a modified version of the model is investigated, testing Hypothesis 3.3.3. This approach changes the linking of the trains and shifts. In the standard model, each move of a train has a *chosenshift* variable that denotes the shift in which the move is being executed, and a *chosenposition* variable denoting the position in the shift that executes the drive/task.

The trains are linked to the shifts, but the shifts are not linked to the trains. In an alternative modeling, the shifts can also be linked to the trains, creating a bidirectional connection between the trains and shifts. As stated in Hypothesis 3.3.3, this modified model could have a higher complexity and worse runtime. With the alternative modeling, the *chosenposition* variable is removed. Additionally, each task in a shift has a *tchosendrive* variable which denotes the train that is driven in the task and *mchosendrive* which move of the train is being executed in this task. The formal definition of this new model with the changed variables and constraints can be found in Section 3.2.7. The experiments compare the runtime of the two models on all instances. As solver, CP-SAT with free search is used, and the same five seeds are used throughout all experiments to ensure comparable results. Each seed is run five times, amounting to 25 runs per instance.

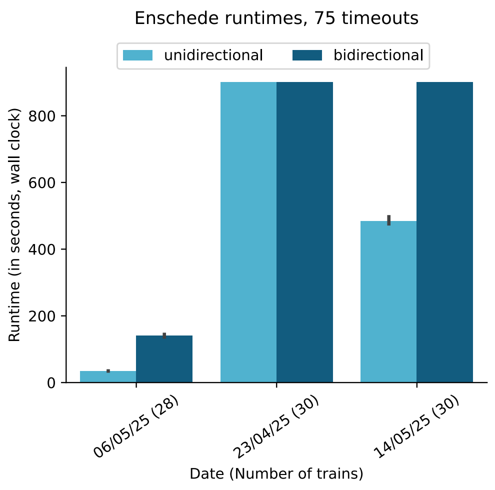
As can be seen in Figure 4.8a, the modified model results in a notable slowdown for all Vlissingen instances. In fact, all runtimes increase by at least a factor of four. The highest difference is observed at the largest instance, where the modified model yields a runtime of $\sim$900 seconds instead of around 125 seconds. Similar observations can also be made for the Enkhuizen station in Figure 4.8b. All the instances achieve an increased runtime for the modified model. The most significant change is again in the largest instance, where the bidirectional model reaches a runtime of 900 seconds or more, and a runtime of 230 seconds without the bidirectional linking. In the Enschede station, depicted in Figure



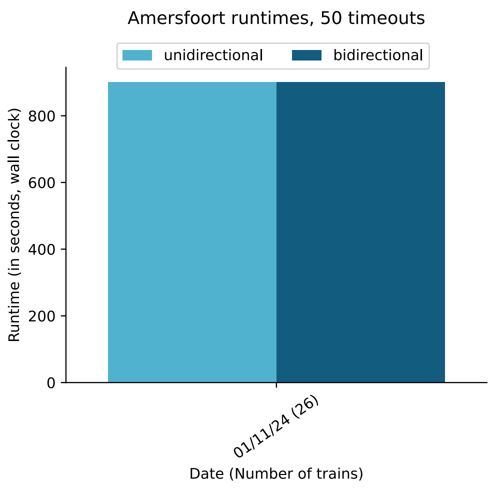
(a) Vlissingen



(b) Enkhuizen



(c) Enschede

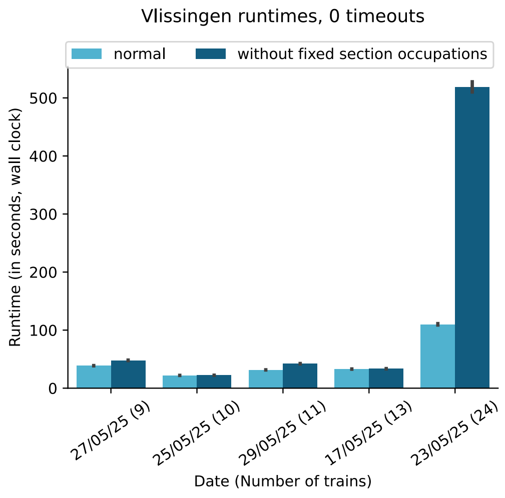


(d) Amersfoort

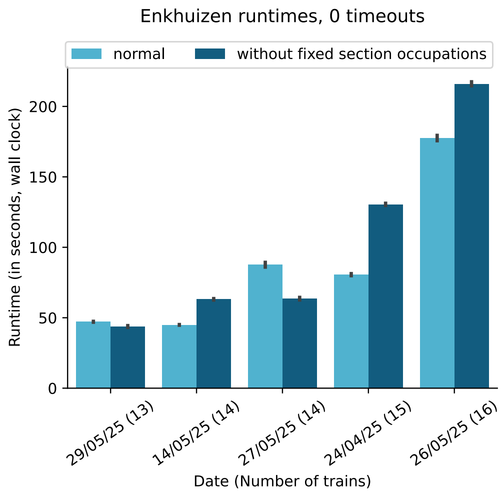
Figure 4.8: Runtimes of model with unidirectional linking ('base model') and bidirectional linking ('modified model')

4.8c, the modified model reaches a slowdown of more than 100 seconds for the instance containing 28 trains, from ∼34 seconds to ∼140 seconds. Both models time out on the 23/04 instance. The 14/05 instance times out as well with the bidirectional linking, compared to ∼ 490 seconds with unidirectional linking. Figure 4.8d shows the runtimes of both models with the Amersfoort station. However, both models result in a timeout after 15 minutes, not allowing for any observations.

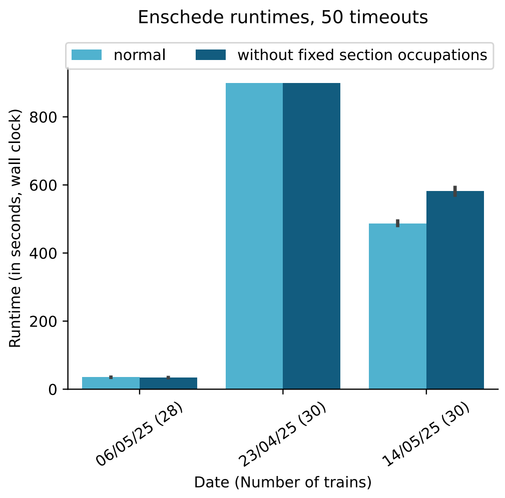
In conclusion, for every instance that did not result in a timeout, the runtime worsened upon the addition of the bidirectional linking. Therefore, it can be said that this modification raises the complexity of the model, leading to a higher computation time. The reason for this could be that since the variables are connected 'twice', when deciding on a value, the solver has to go through a more complex analysis, and thus reach a solution slower. Hypothesis 3.3.3 can be confirmed.



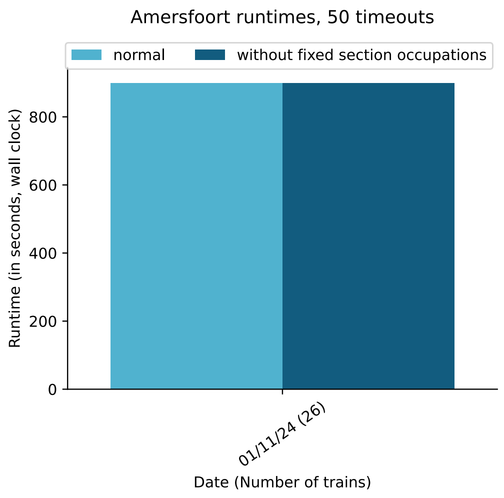
(a) Vlissingen



(b) Enkhuizen



(c) Enschede



(d) Amersfoort

Figure 4.9: Runtimes of the model with and without fixed section occupations

4.7 Removal of Section Occupations

These experiments aim to analyze Hypothesis 3.3.4, investigating if the number of fixed section occupations influences the performance. A fixed section occupation is a section that is used by a through train (does not need to be planned), and therefore occupied and blocked to use for any plannable train. As for example, the smallest Vlissingen instance (27/05, 9 trains) has a high runtime compared to the instances that have more trains (10, 11, and 13 trains), it raises the question whether next to the number of trains there are other attributes that contribute to a high runtime. Since this small instance has the highest number of fixed section occupations (1003), the hypothesis was formed that the runtime could be dependent on the fixed section occupations. Therefore, every instance is run with and without the fixed section occupations of fixed through trains. All the experiments are run with five seeds, each seed five times. The solver used is CP-SAT with free search and a single thread.

The runtimes for the Vlissingen station with and without fixed section occupations can be seen in Figure 4.9a. No instance performs better without the fixed section occupations. For the instances with the most section occupations (27/05 and 23/05), the removal of fixed section occupations yields a worse performance. The 23/05 instance has the largest difference, without section occupations, the runtime increases from 110 seconds to around 500 seconds.

In Figure 4.9b, the plot for the Enkhuizen station is depicted. The results are ambiguous. For the station with the highest amount of section occupations (26/05, 16 trains, 730 section occupations) and two other instances (14/05, 24/04), the runtime worsens. The remaining two instances that result in a lower runtime upon removing the fixed section occupations are the instances with the lowest amount of fixed section occupations, 29/05 (13 trains, 488 section occupations) and 27/05 (14 trains, 499 section occupations).

The Enschede station, depicted in Figure 4.9c does not show any significant improvement or worsening on the 06/05 instance, and the 23/04 instance times out with and without the section occupations. The 14/05 instance worsens its performance when the fixed section occupations are removed. Figure 4.9d for Amersfoort shows that the instance times out on both versions of the model, not yielding results.

Overall, it does not appear that a high number of section occupations causes a worse performance. In most cases, when the fixed section occupations are removed, the runtimes increase. One possible reason could be that through the removal of the occupations, the trains have more possibilities for when to move. This increased search space could lead to a longer search, as more solutions are considered until a feasible one is found. Therefore, Hypothesis 3.3.4 cannot be confirmed.

4.8 Comparing Variable and Value Selection Strategies

In order to guide the solver, variable and value selection strategies can be applied to the search. A variable selection strategy decides which variable is the next to be assigned a value. This could, for example, be the variable with the smallest domain, or the smallest possible value from its domain. A value selection strategy decides the order in which possible values are assigned to certain variables. That way, the solver could try to assign feasible values earlier, resulting in a more efficient runtime. All of these experiments are run with the OR-Tools solver CP-SAT using free search and a single thread; and use the same five seeds, each seed is run five times, resulting in 25 runs per instance.

4.8.1 Walk and Wait Variables

In this section, Hypothesis 3.3.5 about the walk and wait variables is tested. These variables, called *walk* variables in the following, are composed of the time it takes the driver to walk from one location to the next, and the time the driver has to wait to execute the next task. Therefore, this variable denotes the time span from the end of a task to the beginning of the next one (except for at the beginning or end of a shift). If a high value is assigned to a walk variable, it could be that the solver is exploring a solution where a driver has a long waiting time in between tasks. However, during this waiting time, no other task can be executed by the driver, leading to a high probability of currently exploring an infeasible assignment (as other tasks that have to be executed in that time frame will not have a driver available). Therefore, assigning smaller values to this variable could lead the solver to explore satisfiable solutions earlier. The value selection strategy *indomain_min* assigns the smallest possible value; and *indomain_split* bisects the domain, removing the upper half first. As variable selection, *smallest* and *largest*, where the variable with the smallest/largest value in the domain gets selected, and *first_fail* and *anti_first_fail*, where the variable with the smallest/largest domain gets chosen, are used. The instances are therefore run with the following strategies:

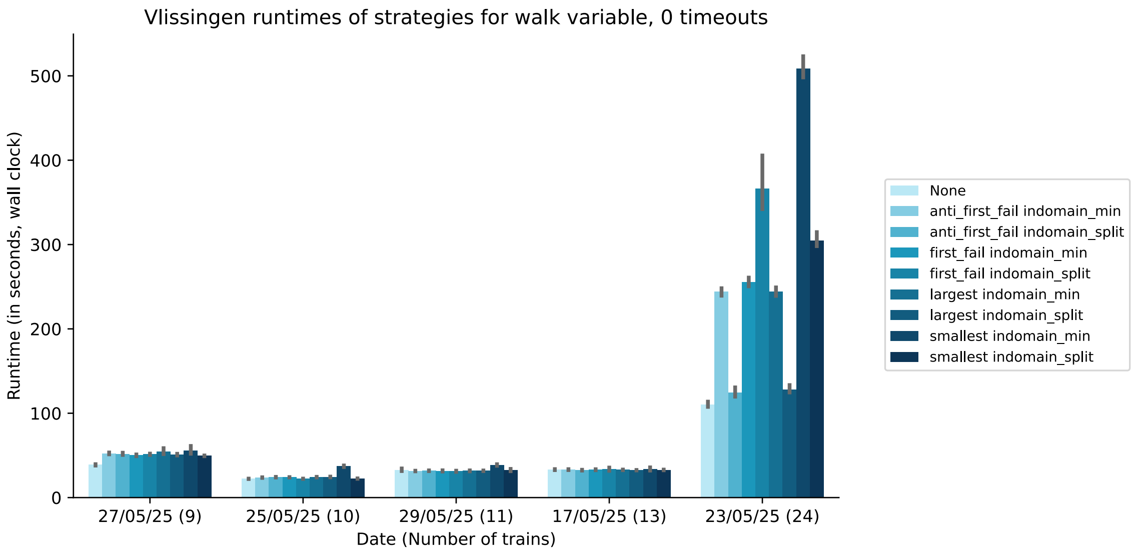


Figure 4.10: Comparison of strategies for the walk and wait time of the drivers for Vlissingen

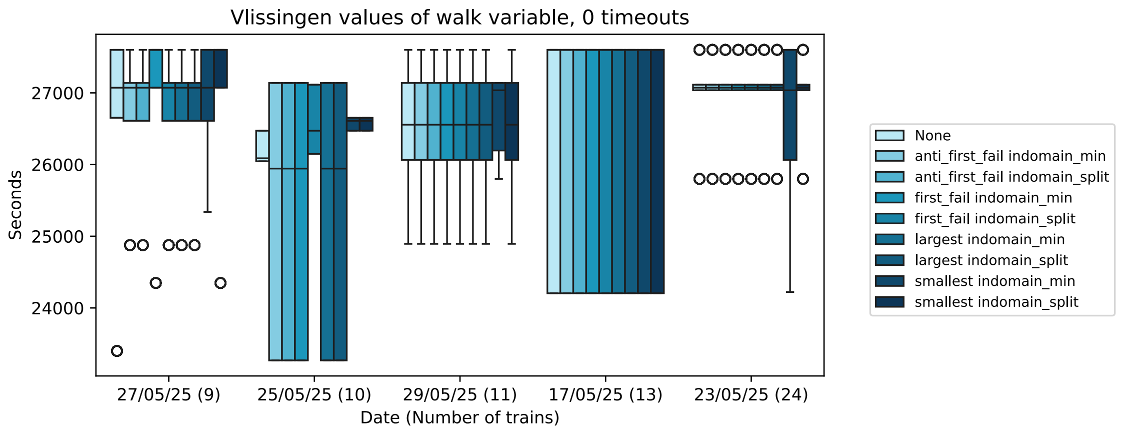


Figure 4.11: Comparison of values for the walk and wait time of the drivers for Vlissingen

Variable selection:	smallest	Value selection:	indomain_split
	first_fail		indomain_min
	largest		
	anti_first_fail		

For each station, there are two plots. One plot displays the runtimes of the model, and the other depicts the assigned values to the walk variables. For each instance, it shows a box plot containing all values of the walk variables, thus the time that passes between two tasks. The box plot displays the quartiles of the values, meaning, for example the lower 25% of the values are in the area from the lowest line to the second-lowest line. The circles describe outliers. This allows an analysis of whether the use of a strategy alters the obtained solution. It is not expected for these values to change significantly, as the duration of the shifts and tasks is fixed, and the overall waiting time just depends on if some routes are taken or not.

In Figure 4.10, the runtimes for the Vlissingen station are depicted. In none of the used strategies, the performance improves. For some of the instances and strategies, the runtimes remain the same, and for others, the runtimes worsen. The assigned values to the walk variable are depicted in Figure 4.11. In the 27/05 and 25/05 instances, the summarized walking and waiting time is indeed lower for most strategies, however, this does not lead to a better runtime. The remaining instances do not have significantly different solutions when strategies are used.

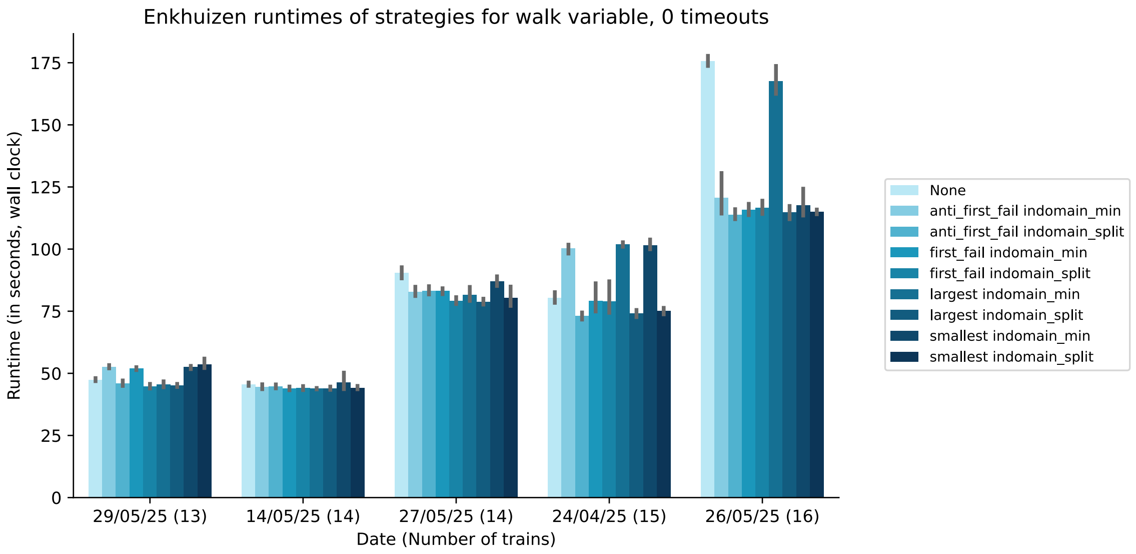


Figure 4.12: Comparison of strategies for the walk and wait time of the drivers for Enkhuizen

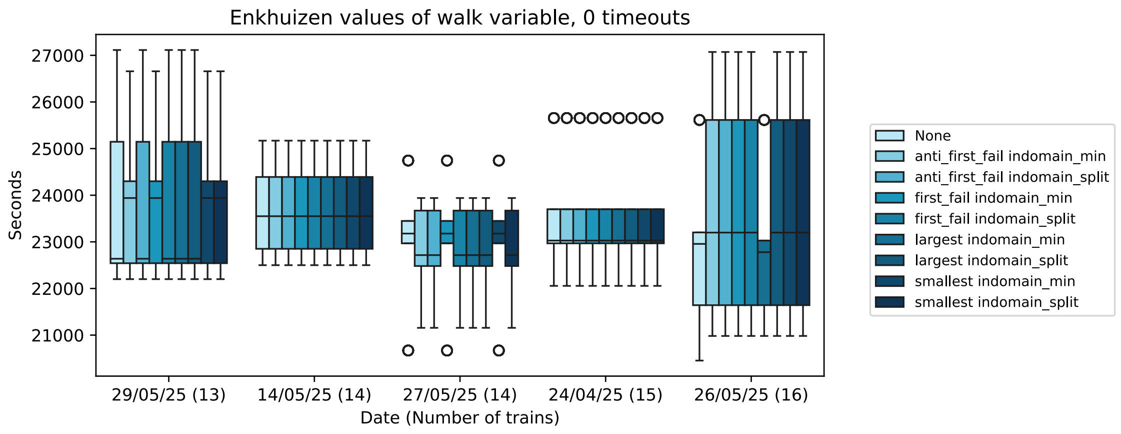


Figure 4.13: Comparison of values for the walk and wait time of the drivers for Enkhuizen

For the Enkhuizen instances, depicted in 4.12, the strategies *anti_first_fail indomain_split* and *largest indomain_split* yield a speed-up for all instances. The greatest improvement is found in the largest instance, where the runtime goes from 175 seconds to around 110 seconds for both aforementioned strategies. The assigned values, shown in Figure 4.13, show that in the improved cases the solution remains unchanged with a strategy in all instances except for 26/05, where the times are even higher than with no strategy. The strategies seem to improve the runtime, but do not necessarily shift the results towards lower values.

The results for the Enschede station are detailed in Figure 4.14. The 06/05 (28 trains) instance does not show any significant improvement using the strategies, and the assigned values, depicted in Figure 4.15, do not differ as well. The 23/04 instance always times out with and without strategies. Interestingly, for the 14/05 instance, all strategies using *indomain_split* improve the performance, while all strategies using *indomain_min* increase the runtime. This could be because there could be no feasible solution where the drivers have no waiting time before a task, because for example, the train is not ready to be shunted to the yard. Then, the assignment to the smallest value could be infeasible, prompting the solver to backtrack. When a value from the lower half of the domain is selected (*indomain_split*), the solver still assigns low waiting times but is less strict, possibly choosing a feasible solution ear-

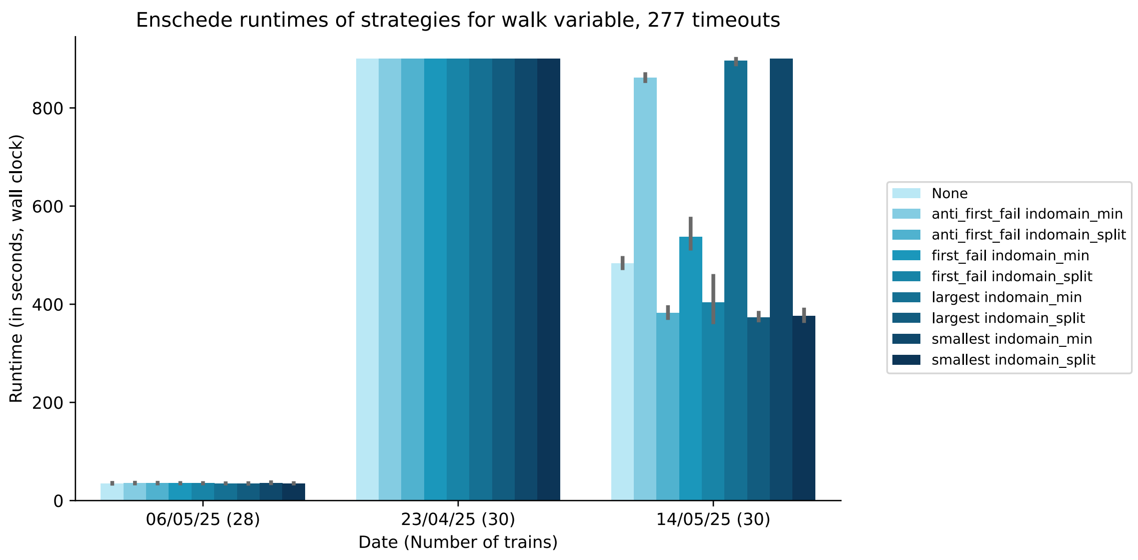


Figure 4.14: Comparison of strategies for the walk and wait time of the drivers for Enschede

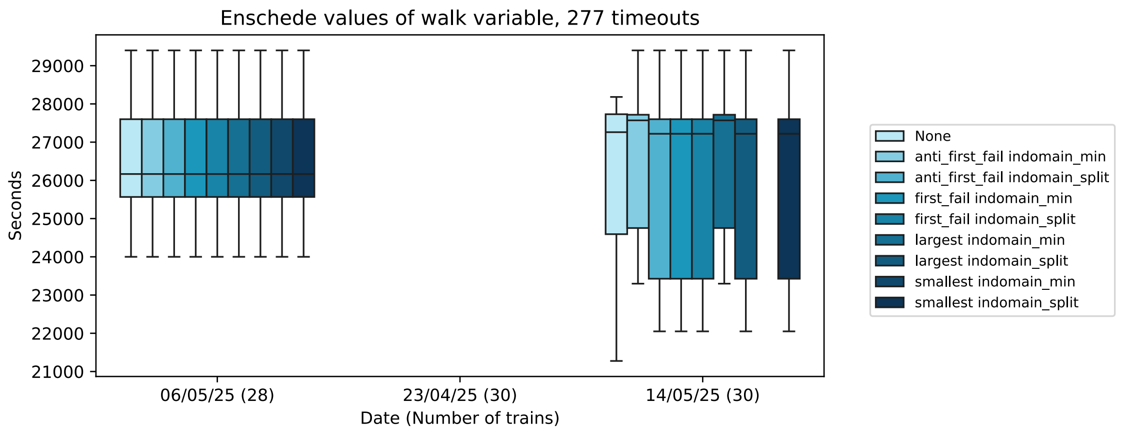


Figure 4.15: Comparison of values for the walk and wait time of the drivers for Enschede

lier. Looking at the assigned values of the walk variables, it can be seen that the assigned values differ slightly. When the runtime improves, the median remains unchanged, but the values assigned are a bit higher.

Overall, the value selection strategy *indomain_split* works well for most instances of Enkhuizen and Enschede. Vlissingen however, did not benefit from any strategy in any instance, which shows that the difference of the stations' layout should be taken into account. As expected, even in cases where runtime improves, the assigned values do not necessarily change towards the lower spectrum. This is because, as the shifts and tasks are fixed, there is not much variation possible in the solution. What is changed is the order in which values are assigned, and as high values can lead to an infeasible solution that has to be backtracked, assigning smaller values first can improve the runtime, namely for Enkhuizen and Enschede.

4.8.2 Variables for Number of Tasks in Schedule (numD)

The numD variable in the model describes the number of tasks (or drives) in a certain shift. The possible number of tasks in a shift is very high, as it is the sum of all possible drives of each train. It is

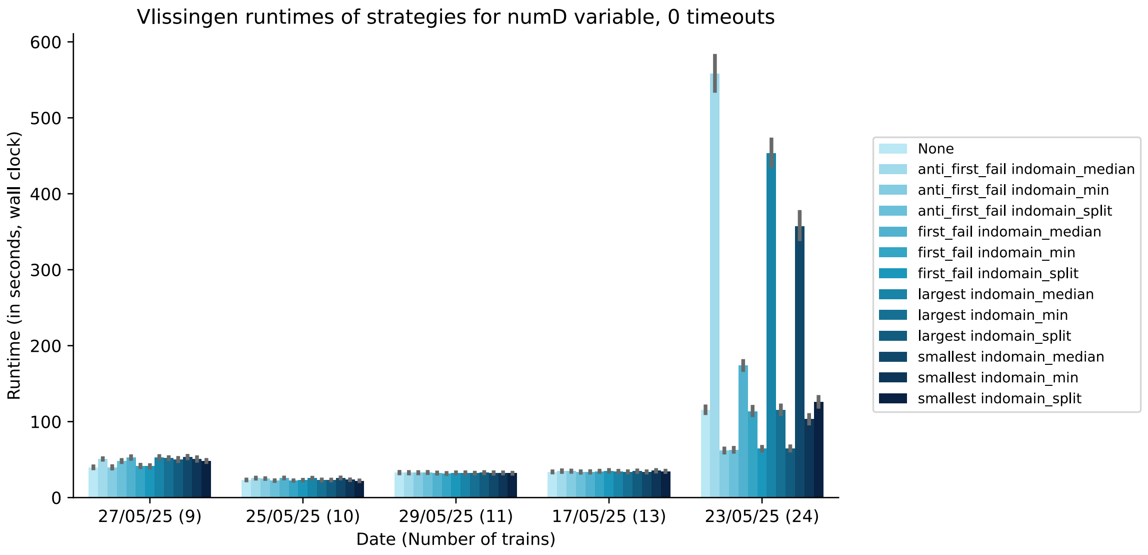


Figure 4.16: Comparison of strategies for the number of tasks in a shift for Vlissingen

unlikely that there are that many tasks in a schedule, as it would mean that one driver executes all tasks. Therefore, one could try to assign smaller values or the median value to this variable, as described in Hypothesis 3.3.6. The strategies *indomain_split*, which selects a value from the lower half of the domain, *indomain_min*, which selects the lowest value in the domain, and *indomain_median*, which selects the median value in the domain, are used. As for the variable selection, the variable with the smallest domain (*first_fail*) and the smallest value in the domain (*smallest*), and the largest domain (*anti_first_fail*) and the largest value in the domain (*largest*) are selected.

Variable selection:	first_fail	Value selection:	indomain_split
	smallest		indomain_min
	anti_first_fail		indomain_median
	largest		

For each station, a plot detailing the runtimes of the individual instances with the corresponding strategies is shown. To further analyze the results, the values assigned to the numD variables are depicted in a plot as well. As for most instances, the numD values remain unchanged upon using a strategy, the plot only displays the instances with a change in the solution. On the y-axis, a box plot displaying the assigned values for the numD variable is shown.

For the Vlissingen station, shown in Figure 4.16, the strategies *anti_first_fail indomain_min*, *anti_first_fail indomain_split*, *first_fail indomain_split*, and *largest indomain_split* are able to halve the runtime for the largest instance (23/05) from around 110 seconds to ∼55 seconds. Interestingly, the values assigned to the numD variable are the same with and without strategy. This could mean that different solutions are not found, the solver is just coming across the solution earlier. For the remaining instances, these strategies for the numD variable do not seem to have a significant influence on the runtime, however, the values do change with a strategy, as can be seen in Figure 4.19. For example, in the 17/05 instance, where with no strategy one shift performs all tasks and with strategies, the tasks are more distributed equally. Overall, the strategies *anti_first_fail indomain_min* and *first_fail indomain_split* either do not yield a significant change or result in a speed-up in all instances. The strategies using *indomain_median* perform the worst. This could support the claim that most of the bigger values, and even the median value in the domain of numD lead to infeasible solutions.

The results for Enkhuizen are depicted in Figure 4.17. For the four smaller instances 29/05, 14/05,

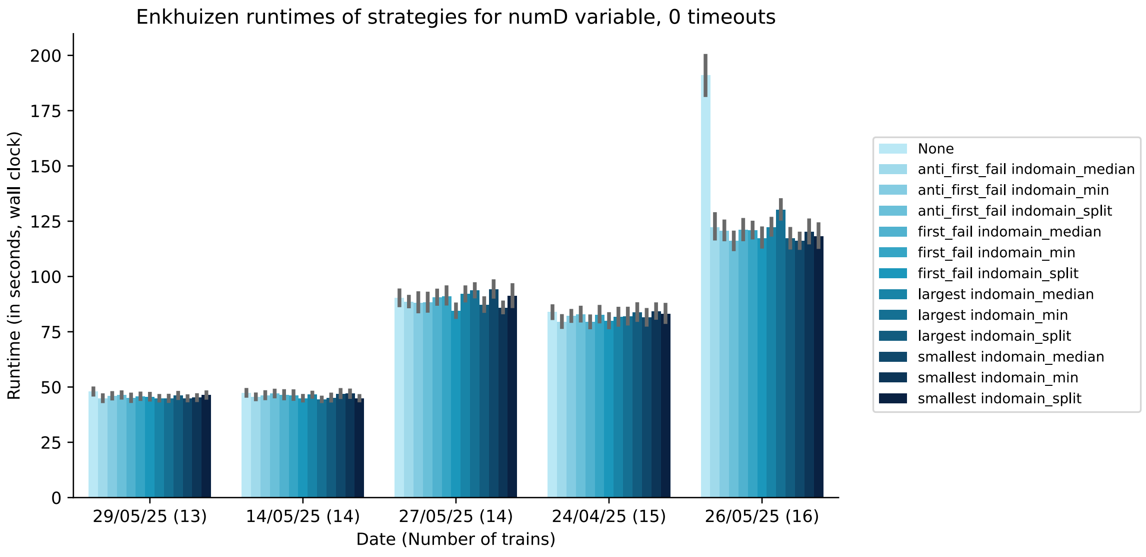


Figure 4.17: Comparison of strategies for the number of tasks in a shift for Enkhuizen

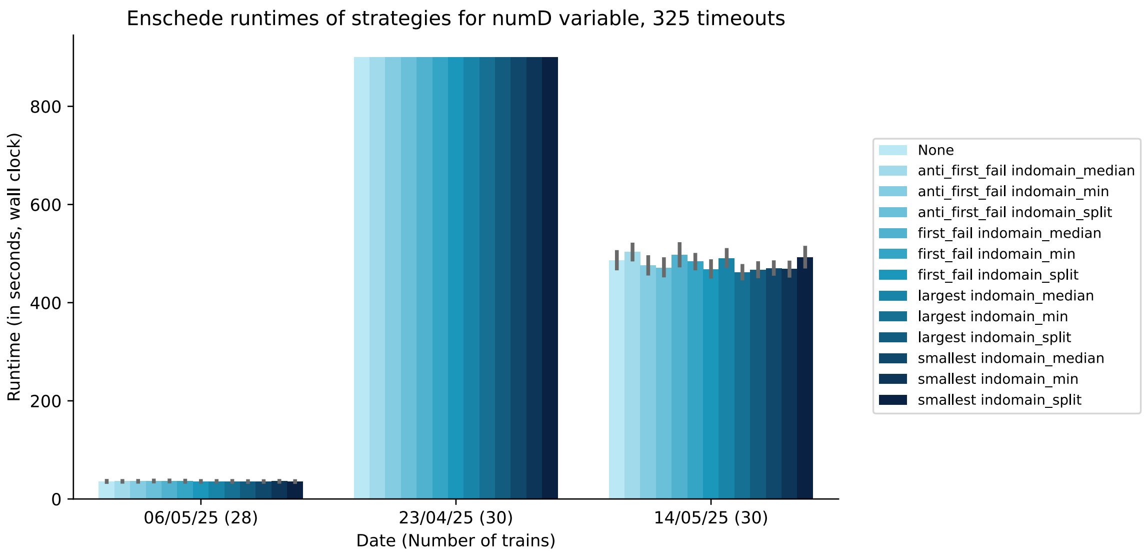


Figure 4.18: Comparison of strategies for the number of tasks in a shift for Enschede

27/05, and 24/04, the strategies for the numD variable do not seem to particularly help the solver to achieve a lower runtime. The strategies result in slightly faster runtimes, but not by a large margin. The assigned values for numD also remain unchanged, except for 29/05, where using a strategy does result in lower values for numD, shown in Figure 4.19. The largest instance however, does profit from using any strategy, reducing the runtime from almost 200 seconds to $\sim$125 seconds when employing a strategy, no matter which one. The values of the numD variables change as well with a strategy, as can be seen in Figure 4.19, the numD variables take lower values.

In the Enschede station, depicted in Figure 4.18, the 23/04 instance times out with and without strategies, and no conclusions can be drawn. The 06/05 and 14/05 instances do not seem to generally profit from using a strategy, as the runtimes do not change substantially. Moreover, the strategies do not seem to have an influence on the values assigned to the numD variables, as they remain unchanged.

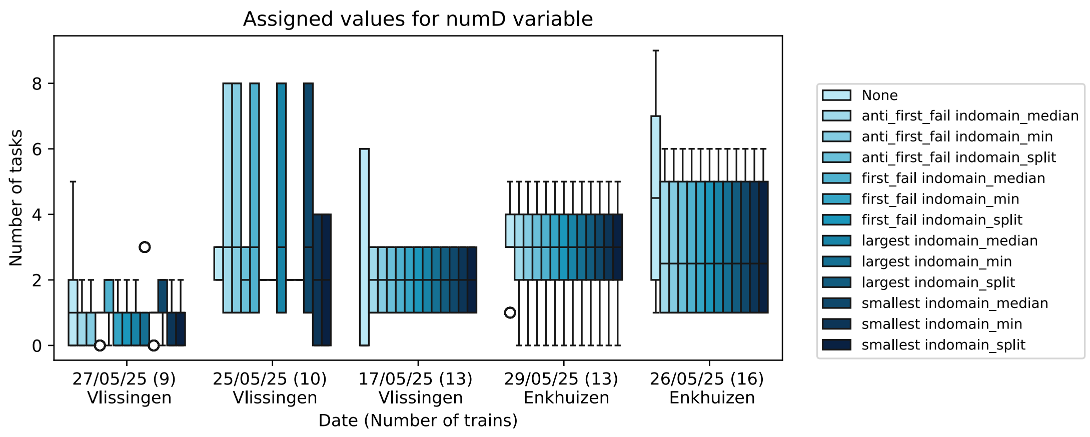


Figure 4.19: Comparison of values for the number of tasks in a shift

In conclusion, using strategies for the number of tasks in a schedule does not influence the runtime in most cases, however, in some specific cases it can be very beneficial. The most promising strategies seem to be *anti_first_fail indomain_min* and *first_fail indomain_split*, as they either lowered the runtime or stayed approximately the same for all instances. The improvement with the *indomain_min* and *indomain_split* strategies can be attributed to the fact that high values are not assigned, because a value that is from the lower half of the domain or the smallest value in the domain is assigned. As the domain is large and the probability of a high value being part of a solution is low (because it would mean that one shift executes all the tasks), the selection strategies are guided to not take these (probably infeasible) values, thus reaching a solution faster. This would also explain the poor performance of the *indomain_median* strategy, the median value is still very high and thus unlikely to be in a feasible solution. Hypothesis 3.3.6 can be partly confirmed, using a strategy on the number of tasks in a shift can yield a lower runtime.

4.8.3 Chosenshift Variable

These experiments test the Hypothesis 3.3.7. Every movement of a train has a chosenshift variable. It describes the shift in which this move is being executed. In the following experiments, the solver tries to assign the next available shift (in order of appearance) that still has time for this new task. This could lead to a better performance as the solver does not try to assign a 'random' shift. For this, *indomain_split* and *indomain_min* will be used. The *indomain_split* strategy halves the domain of the variable, only assigning values of the lower half. With *indomain_min*, the solver will always take the lowest value in the current domain, therefore filling the shifts up one after another. As variable selection, *first_fail* and *smallest* will be used. *first_fail* selects a variable with the smallest domain, and *smallest* selects a variable with the smallest value in the domain.

Variable selection:	first_fail	Value selection:	indomain_split
	smallest		indomain_min

For all instances except four, the values for the chosenshift have the same distribution. These four exceptions are depicted in Figure 4.23, where the distributions of the number of tasks in each shift are shown. That way, it can be reconstructed if the strategies result in different values regarding the chosenshift. As a value from the lower half of the domain or the lowest is selected, it is interesting to see if the 'lower' shifts contain more tasks. The plot can be read the following way: On the x-axis, the instances, subdivided by the strategies, are depicted. The y-axis shows the different shifts. Every shift is assigned a number. The size of the colored area describes the amount of tasks in shift number y. If the colored areas are bigger at the 'lower' shifts, for instance, it means that smaller values were chosen for the chosenshift variable.

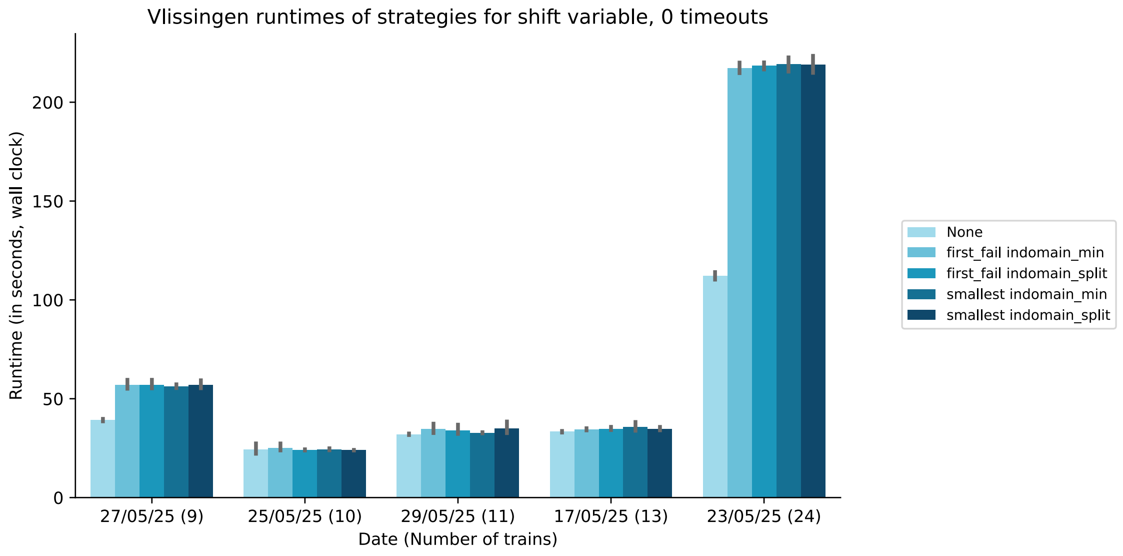


Figure 4.20: Comparison of strategies for the chosenshift variable for Vlissingen

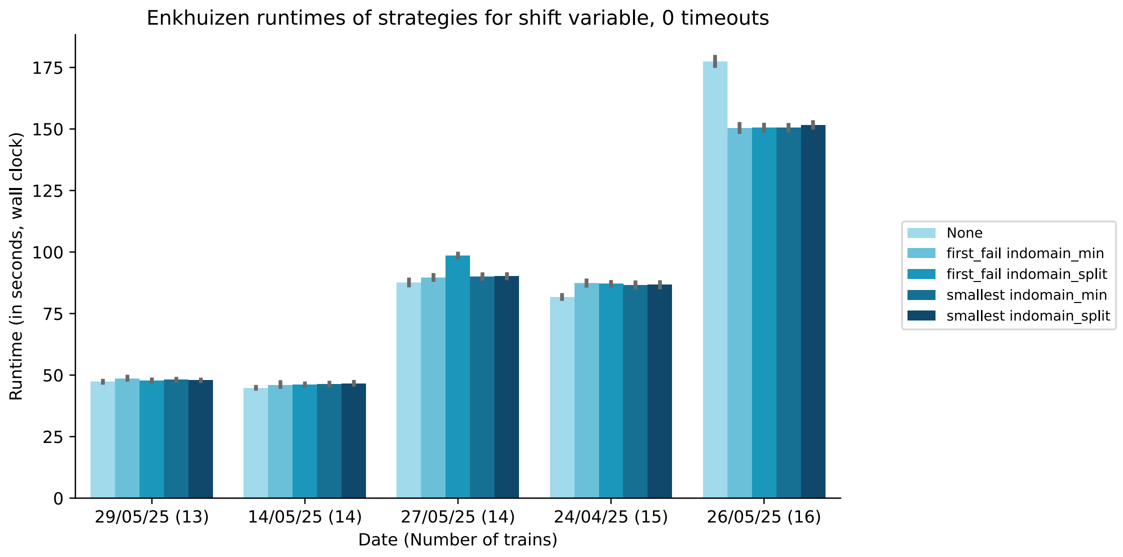


Figure 4.21: Comparison of strategies for the chosenshift variable for Enkhuizen

For the Vlissingen station, depicted in Figure 4.20, the strategies do not improve performance. In fact, for the instances with 9 and 24 trains, the runtimes increase with all tested strategies. For the 27/05 (9 trains) instance, using any strategy will also change the chosen shifts towards shifts with a lower number, as seen in Figure 4.23. In the instance with 24 trains, the solution for the chosenshift variables is the same with and without a strategy, using a strategy just slows down the finding of this solution. The reason for this could be that the tasks cannot be applied in a manner where 'lower' shifts get more tasks, and therefore, the solver has to go through many infeasible solutions and loses time there. In the 17/05 instance, using a strategy gives a different solution with lower shifts, while using no strategy results in shift 1 having all tasks in their schedule. This change in the solution however, does not influence the runtime. The remaining instances do not have a change in performance depending on the strategy, and the assigned values are also the same.

For Enkhuizen, depicted in Figure 4.21, the three smallest instances (29/05, 14/05, and 27/05) do

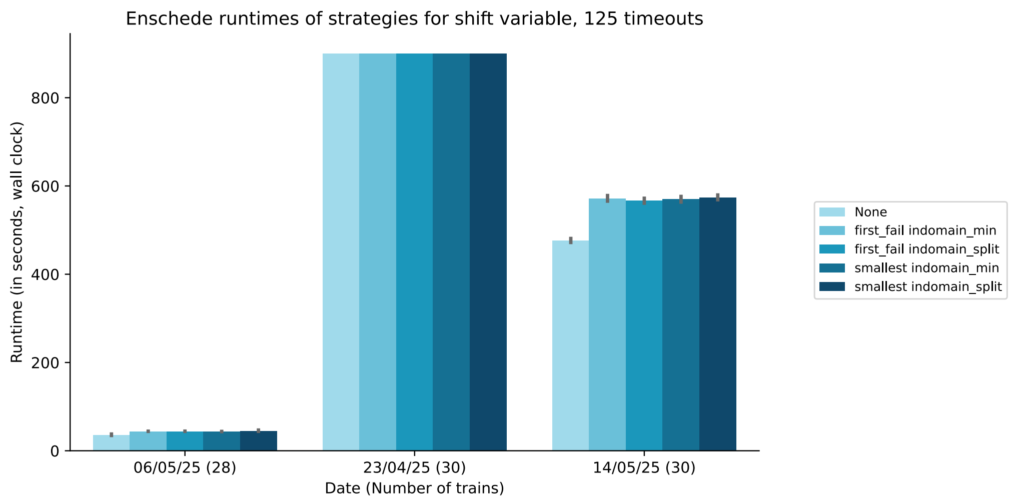


Figure 4.22: Comparison of strategies for the chosenshift variable for Enschede

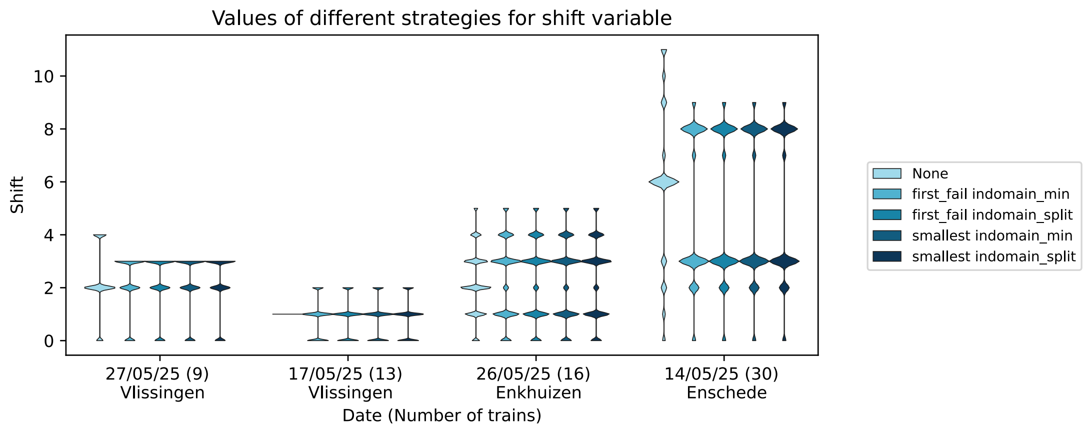


Figure 4.23: Comparison of values for the chosenshift variable

not result in a different runtime when using a strategy. Subsequently, the values of the solutions also do not differ. Using a strategy with the 24/04 instance results in a slightly worse performance while having the same values in the solution. In the largest instance, however the use of strategies is beneficial: Using a strategy reduces the runtime from around 180 seconds to around 150 seconds for all strategies. It also shows a change in the solution, depicted in Figure 4.23: 'lower' shifts have more tasks. Therefore, assigning the lowest possible shift (or from the lower half of possible shifts) seems beneficial.

For the small Enschede instance, shown in Figure 4.22, the variable and value selection strategies increase the runtime from around 35 seconds to around 44 seconds with any strategy. The values for the number of tasks in a shift, however, remain the same. For the 23/04 instance, each selected strategy times out, not allowing further analysis. The 14/05 instance also worsens with the use of a strategy, and the chosen shifts are more distributed in the lower shifts, as can be seen in Figure 4.23.

Interestingly, the different strategies used all give the same solutions in every instance. This could be attributed to the similarity of the strategies (assigning low values) and a small search space, as there could be just a few different combinations of how to distribute the shifts. For example, in the shift distribution for the 27/05 Vlissingen instance, depicted in Figure 4.24, it can be seen that shift 1 does not overlap with any other shift. Therefore, every task that has to be executed in the timeframe of shift 1 has to be executed by shift 1, and another shift cannot be chosen. Overall, using a strategy only improved the performance of one instance, and worsened or did not change in the remaining instances.

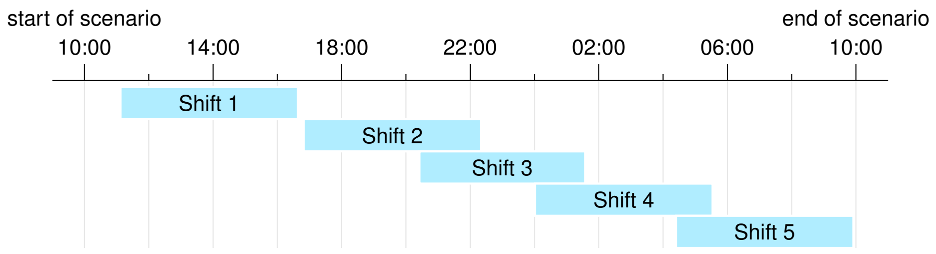


Figure 4.24: Distribution of shifts for the 27/05 Vlissingen instance

The Hypothesis 3.3.7 does not seem to be confirmed, guiding the selection of the shift does not reduce the runtime. The strategies could be unsuccessful because of the unidirectional linking. As the train variables link to the shift variables but not vice versa, starting by assigning the chosenshift variables could lead to increased backtracking. This is because the chosenshift variables could be assigned before the times for the routes of the trains are assigned. When subsequently the times and routes for the trains are assigned, the chosen chosenshift variables could turn out to be infeasible, making the search longer and increasing the runtime.

4.8.4 Variables for Duration of Stay on the Departing Platform

In these experiments, search strategies are imposed on the start of the parking on the departure platform, testing Hypothesis 3.3.8. The strategies are applied to the variable that denotes the start time of the last stay, which is the stay at the platform from which a train departs. Assigning a higher value means that the train arrives later at the departing platform, and thus the time on the platform is minimized. This could be of interest, as a long time on the platform could mean that the train is blocking another one, thus more likely to result in an infeasible assignment. These experiments were also conducted by Gincheva. In Gincheva's experiments, the strategies turned out to work very well for Enkhuizen, where the runtime was halved with *SelectMaxValue* and even approximately 5 times faster for *SelectUpperHalf*. However, these results did not recur with Amersfoort, where there was no notable speed-up. The strategies employed are the same ones that are tested by Gincheva, *SelectMaxValue (indomain_max in Minizinc)* and *SelectUpperHalf (indomain_reverse_split in Minizinc)*. As variable selection strategy, the strategies *anti_first_fail*, *first_fail*, *largest* and *smallest* are tested.

Variable selection:	first_fail	Value selection:	indomain_reverse_split
	smallest		indomain_max
	anti_first_fail		
	largest		

Next to a plot of the runtimes for each instance, a plot depicting the assigned values for the stay on the departing platform is shown. For an instance, it shows a box plot containing all values of the duration of the last stay of trains that are departing trains, base shunt trains, or split trains, departing from a platform (not a yard). The box plot displays the quartiles of the values, meaning for example the lower 25% of the values are in the area from the lowest line to the second-lowest line. The circles describe outliers. The lower the values for the durations are, the higher the value on the start variable of the last stay.

In Figure 4.25, the runtimes for the Vlissingen station are depicted. For 3 out of the 5 stations, imposing a search strategy does not yield a different runtime (25/05, 29/05, 17/05). The smallest (27/05) and largest (23/05) instances, however, result in a higher runtime. The largest instance has a significant increase, especially with the *indomain_reverse_split* strategies. As can be seen in Figure 4.26, in most instances the assigned values are lower or remain the same when a search strategy is imposed. For the smallest instance, the duration of the last stay gets higher overall, contrary to what would be expected. This is because with the selection strategies, many trains that arrive and depart at the same platform are

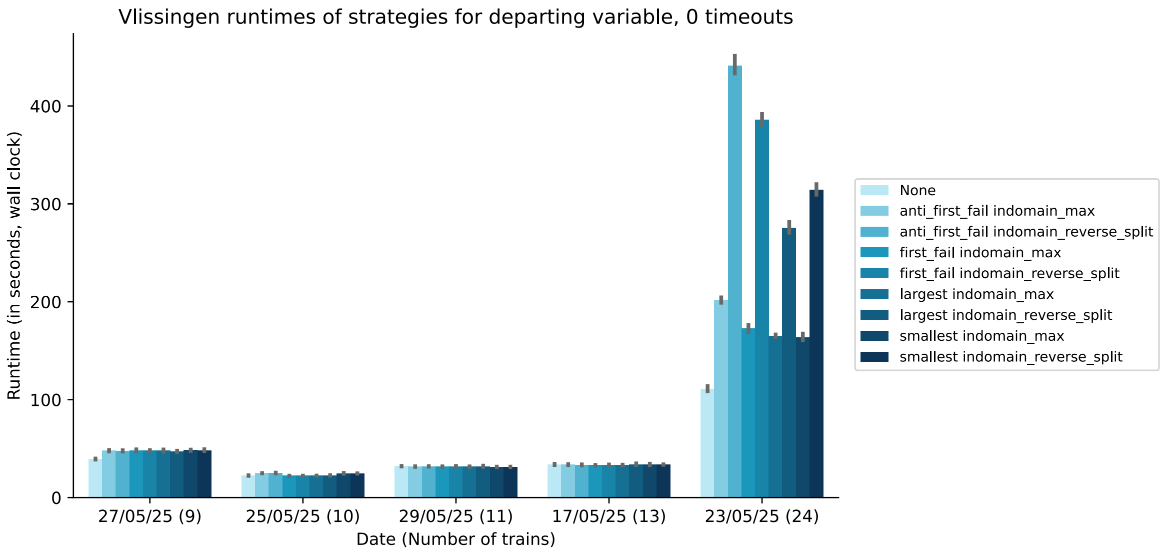


Figure 4.25: Comparison of strategies for the duration on the departing platform for Vlissingen

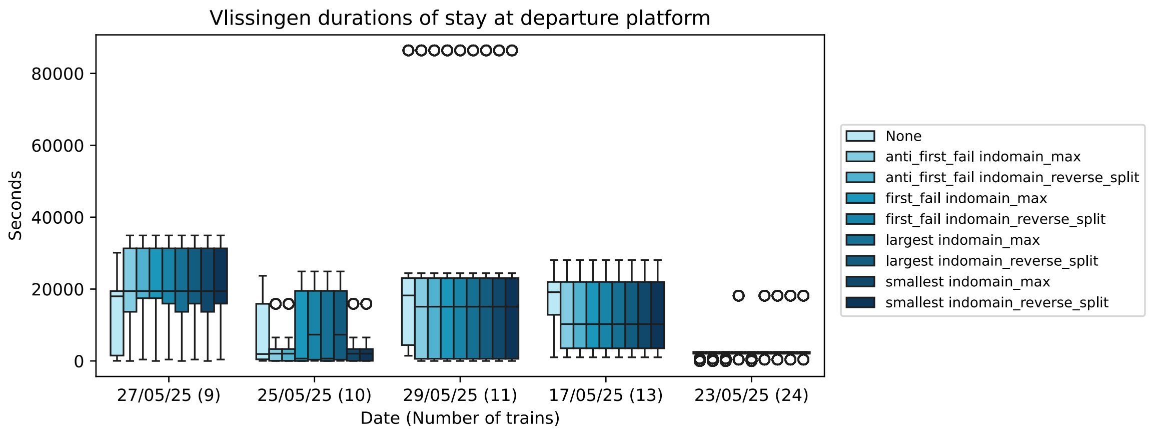


Figure 4.26: Comparison of values for the duration on the departing platform for Vlissingen

not assigned to go to a yard and come back, but rather stay at that platform. Through this, the only possible value for the start time of the last stay is the arriving time, resulting in a high duration. The reason for this assignment could be that if the route gets chosen where the train stays at the platform (no route), then the domain only consists of one value, and the stay on the last platform cannot be minimized.

The runtimes for the Enkhuizen station can be seen in Figure 4.27. The instances 29/05 (13 trains) and 14/05 (14 trains) do not profit from a search strategy. The remaining instances improve upon using a search strategy. The most notable speed-up is seen in the largest instance, where the runtimes decrease from ∼175 seconds with no strategy to ∼110 seconds with the strategies *anti_first_fail indomain_reverse_split and indomain_min*,*largest indomain_max* and*smallest indomain_reverse_split*. For the four smaller instances, the use of strategies results in a lower duration of the stay at the platform, as can be seen in Figure 4.28. There are a few outliers, for example, in the 14/05 instance, but interestingly, these do not result in a different runtime. The assigned values of the largest instance, however, do influence the runtime: for all improved runtimes, the durations of the stays are larger. The reason for this could be, as in Vlissingen, that while many trains have a stay duration of 0, there are some that stay at the platform and never get routed to the yard, resulting in a high duration on the platform.

The Enschede station, depicted in Figure 4.29, does not improve upon using a search strategy. The

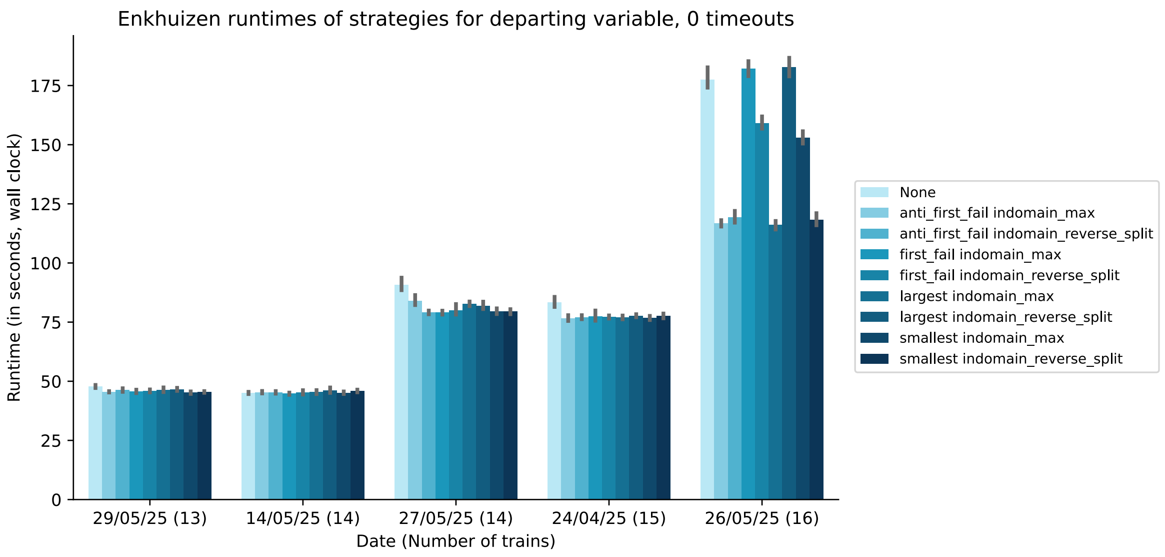


Figure 4.27: Comparison of strategies for the duration on the departing platform for Enkhuizen

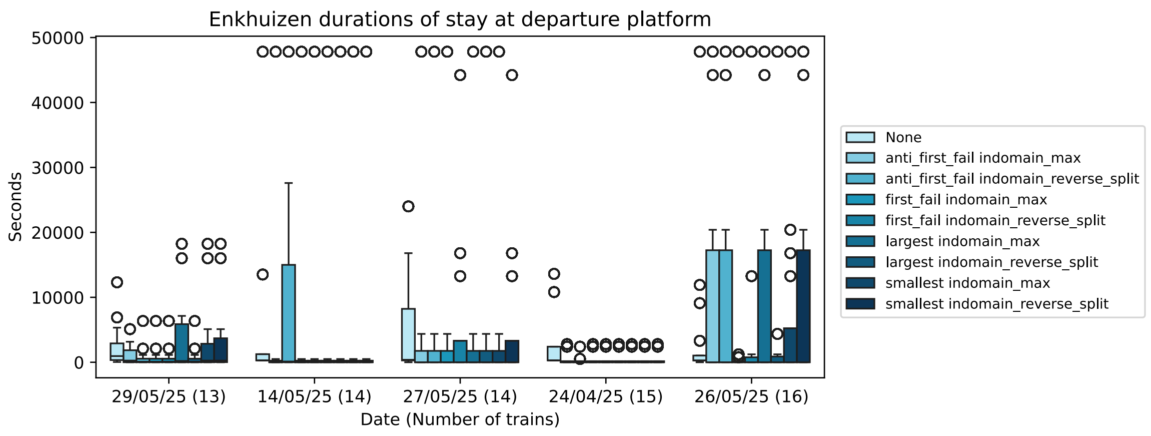


Figure 4.28: Comparison of values for the duration on the departing platform for Enkhuizen

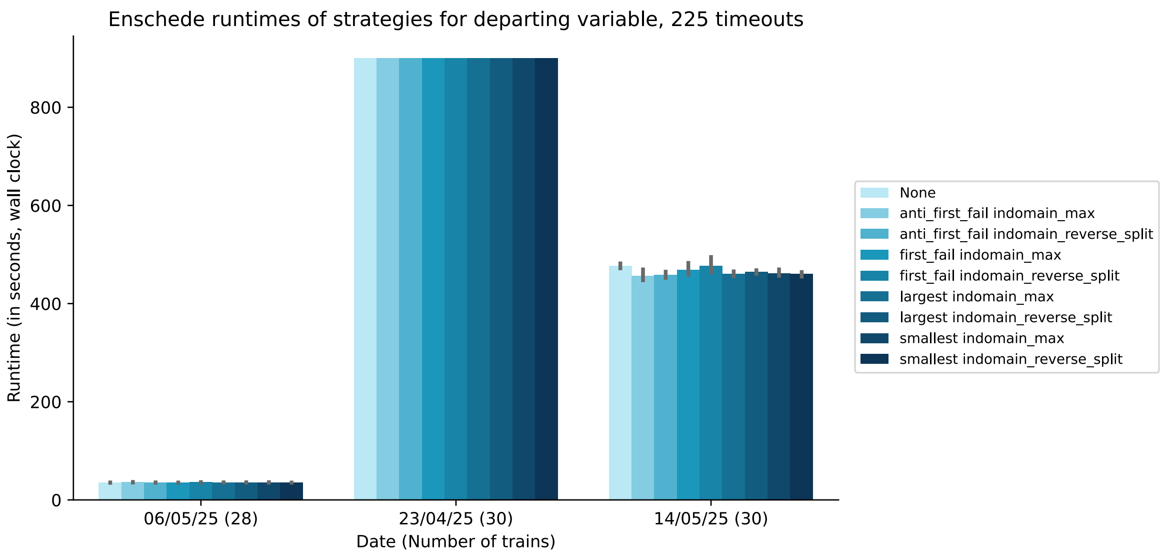


Figure 4.29: Comparison of strategies for the duration on the departing platform for Enschede

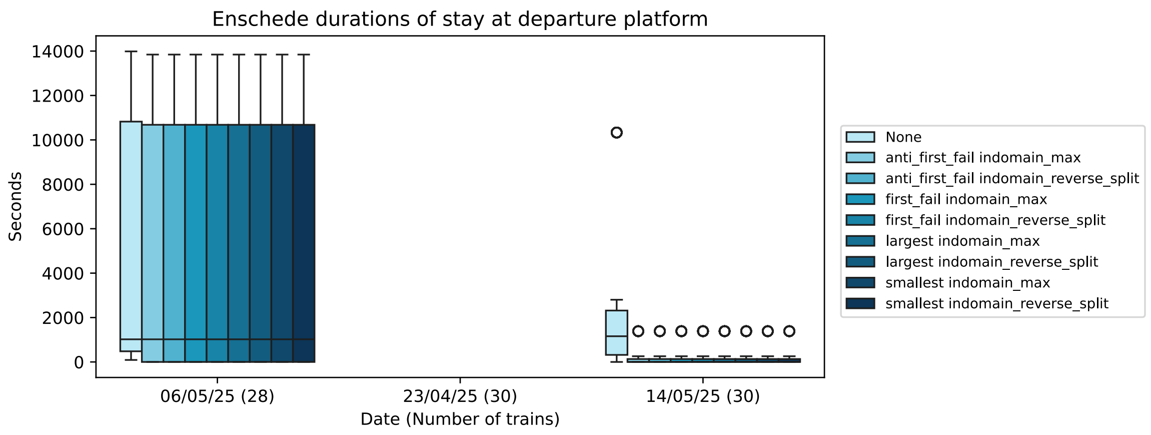


Figure 4.30: Comparison of values for the duration on the departing platform for Enschede

23/04 instance times out with and without a strategy, and the 06/05 and 14/05 instances have similar runtimes when using a strategy. Even though the runtimes do not differ, the assigned values are lower when a strategy is employed, as can be seen in Figure 4.30.

Overall, there are no strategies that obtain a speed-up on each instance. The Vlissingen station performs worse with strategies, and Enschede does not result in different runtimes. Only Enkhuizen improves upon using strategies for the stay at the departing platform. The observations made by Gincheva - that only Enkhuizen improves - could generally be reproduced. This shows that even with similar attributes, the individual stations do exhibit varying behavior. This varying behavior could be taken into account when designing algorithms for node planning.

4.8.5 Conclusion of Strategies

In general, it seems that the efficiency of employing a variable and value selection strategy is highly dependent on the instance. The results across individual instances are inconsistent, making it hard to draw conclusions that apply universally. The most promising variable to use a strategy on seems to be the numD variable, which yields a speed-up in some cases, and never worsens for certain strategies. The use of a strategy on the walk and wait variable can reduce the runtime for the Enkhuizen and Enschede stations, but not for Vlissingen. The strategies on the chosenshift variable yield an improvement for only one out of the 14 instances. Gincheva's experiments with strategies on the stay on the departing platform resulted in the finding that the strategies work well only for Enkhuizen, not for Amersfoort. This finding is reproduced: The strategies reduce runtime for Enkhuizen, but not for the remaining stations. Despite the variability, using variable and value selection does seem promising for some cases. Further research with a wider set of instances could help to identify strategies that yield constant improvements.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

In this thesis, the existing CP model for the SRP, developed by Gincheva, is combined to a new model that simultaneously solves the SRP and the drivers' problem. This combined model can be used to generate part of the initial solution for the HIP at NS, which generates a shunt plan for a node. By already including the drivers and their shifts in the initial solution, it avoids the need to recompute the routes in case the shunt plan cannot be performed with the number of drivers available. This could reduce the computation time of the local search. The model can also be used outside of the HIP. In the case of a modified train schedule, it can verify whether the available drivers are sufficient to carry out all shunting tasks without having to run the local search, which can take several hours.

The use of Gincheva's SRP model in Minizinc required preliminary work. The model was extended to function at additional stations, and the preprocessing was automated to quickly generate new instances by directly identifying trains that need to be removed. Split and combined trains, which were absent in the original MiniZinc model, were also added. Furthermore, Gincheva's original analysis was expanded. While her work tested only one instance for Enkhuizen and one for Amersfoort, this thesis evaluated 14 instances across four stations. The results reproduced and confirmed her main findings. Building on this groundwork, a new model was developed that combines the drivers' problem with the SRP, producing schedules for drivers alongside the routes for trains. For each driver's shift, the model correctly computes a schedule of routing trains to the yard and back, taking into consideration the walking times between tasks. At NS, it is aimed to solve the problem in a few minutes for mid-sized stations. The model exhibits a satisfactory runtime for the small and mid-sized stations Vlissingen and Enkhuizen (under 200 seconds for all instances), but higher runtimes for Enschede and Amersfoort. As expected, the addition of the drivers' problem to the SRP increases the complexity and consequently, the runtime. To mitigate these effects, structural modifications and solving strategies are explored.

Using a unidirectional linking of the train and shift variables results in a lower runtime compared to a bidirectional linking. This finding also suggests further improvements may be possible by removing the linking altogether and introducing new combined variables. To guide the solver towards finding a solution faster, several variable and value selection strategies are tested. In particular, strategies targeting the variable that determines the number of tasks in a shift improve computation times for certain instances, and worsen for none. However, the efficiency of all strategies varies significantly between individual instances and stations. For example, Vlissingen often showed no improvement with any strategy. The experiments contribute valuable insights into the value and variable selection strategies, and applying such strategies to other instances at NS could lead to further performance improvements.

The solver analysis reveals that enabling multiple threads and free search significantly improves runtime for the Google OR-Tools solver CP-SAT. Since the HIP algorithm currently uses only one thread, investigating multi-threaded approaches at NS could be beneficial. The solver Chuffed exhibits efficient results as well, and, in certain cases, even outperforms the multi-threaded version of CP-SAT.

The study also shows that station and instance attributes strongly affect computation time. Generally, the more moves can be made by the trains and the more sections a route contains (dependent on the size of the station), the higher the runtime. Fixed section occupations also have an influence: A higher number of sections that are occupied and thus cannot be used for a plannable train does not lead to a higher complexity, but in some cases even to an improved runtime. By leveraging these insights on different station attributes, more targeted enhancements could be incorporated into NS algorithms to improve efficiency at particular stations.

By combining the drivers' problem with the SRP and performing experiments across multiple stations, this thesis delivers a practical model that can be used for shunt planning at NS and demonstrates the applicability of Constraint Programming for scheduling shunting trains and their drivers.

5.2 Future Work

The developed model successfully computes solutions to the SRP and the drivers' problem combined. However, its runtime is considerably higher compared to solving the SRP on its own. To address this, future work is proposed to improve performance and introduce modifications that facilitate more effective analysis.

5.2.1 Expand Test Instances

For the Enkhuizen station, the available instances combined the three yards into one. Experiments could be conducted using the original three yards to determine how complexity varies with the number of yards. In addition, the influence of different train types could be examined, as the available instances contained a limited variety in the type of trains. Overall, testing more instances and stations would allow for a wider analysis of the model's performance and help to draw conclusions about the impact of different station layouts.

5.2.2 Modification of Walk and Wait Variable

The walk and wait variable could be split into two different variables, one for the walking and one for the waiting. That way, either the walking or the waiting time could be specifically minimized, allowing for more detailed analysis.

5.2.3 Search Space and Domains

A way to improve runtime is to reduce the search space. For example, the constraint to shunt every train at most one hour after arrival could be added. This would reduce the possible times the trains could be routed, and thus the search space. With a reduced search space, the solver possibly has to go through fewer assignments to find a feasible one, thus improving computation time.

Moreover, the domain of the shift variables could be reduced. As of now, the domain contains all possible tasks from all trains combined, even those whose execution times do not overlap with the shift. Removing those from the domain could improve performance.

5.2.4 Alternative Modeling

As the experiment of the unidirectional and bidirectional linking showed, additional linking increases the complexity and, therefore the runtime. Thus, the model could be remodeled so as not to use a linking in between the driver and train variables at all, but to combine the variables so that no linking is needed. In this alternate modeling, the driver shifts are then connected to the tasks by a circuit constraint, pointing at the next task. By not having separate variables for the drives of a driver and the moves of a train, the complexity could be reduced.

Bibliography

[1] https://tex.stackexchange.com/questions/554238/how-can-i-draw-a-train-with-tikz. – accessed on 15 February 2025, 16:47

[2] BOURREAU, Eric ; GARAIX, Thierry ; GONDRAN, Matthieu ; LACOMME, Philippe ; TCHERNEV, Nikolay: A constraint-programming based decomposition method for the Generalised Workforce Scheduling and Routing Problem (GWSRP). In: *Int. J. Prod. Res.* 60 (2022), Nr. 4, S. 1265–1283. – URL https://doi.org/10.1080/00207543.2020.1856436

[3] BOYSEN, Nils ; EMDE, Simon ; FLIEDNER, Malte: The basic train makeup problem in shunting yards. In: *OR Spectr.* 38 (2016), Nr. 1, S. 207–233. – URL https://doi.org/10.1007/s00291-015-0412-0

[4] BROEK, Roel van d.: Train shunting and service scheduling: an integrated local search approach. (2016)

[5] BROEK, Roel van den ; HOOGEVEEN, Han ; AKKER, Marjan van den: Personnel Scheduling on Railway Yards. In: HUISMAN, Dennis (Hrsg.) ; ZAROLIAGIS, Christos D. (Hrsg.): *20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2020, September 7-8, 2020, Pisa, Italy (Virtual Conference)* Bd. 85, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, S. 12:1–12:15. – URL https://doi.org/10.4230/OASIcs.ATMOS.2020.12

[6] BROEK, Roel van den ; HOOGEVEEN, Han ; AKKER, Marjan van den ; HUISMAN, Bob: A Local Search Algorithm for Train Unit Shunting with Service Scheduling. In: *Transp. Sci.* 56 (2022), Nr. 1, S. 141–161. – URL https://doi.org/10.1287/trsc.2021.1090

[7] (DHPC) Delft High Performance Computing Centre: *DelftBlue Supercomputer (Phase 2)*. https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2. 2024

[8] EISOLD, Jan ; PREIS, Henning ; BEŠINOVIĆ, Nikola: *Dimensioning of track groups in rail freight stations on the basis of an optimised waiting probability*. https://www.conftool.com/or2024/index.php?page=browseSessions&search=Besinovic. 2024. – accessed on 9 January 2025, 09:49

[9] EUROPEAN COMISSION: *New Action Plan: boosting long-distance and cross-border passenger rail*. https://transport.ec.europa.eu/news-events/news/action-plan-boost-passenger-rail-2021-12-14_en. 2021. – accessed on 15 February 2025, 15:17

[10] GINCHEVA, Nevena: Routing Shunt Trains at NS with Constraint Programming. (2024). – URL https://resolver.tudelft.nl/uuid:0a1bd7e5-5648-4cc3-aa4a-0e177c7e9957

[11] GUEI-HAO CHEN, Jyh-Cherng J. ; HAN, Anthony Fu-Wha: Applying Constraint Programming and Integer Programming to Solve the Crew Scheduling Problem for Railroad Systems: Model Formulation and a Case Study. In: *SAGE journals* 2676 (2022), S. 408–420. – URL https://journals.sagepub.com/doi/epub/10.1177/03611981211036368

[12] HAAHR, Jørgen T. ; LUSBY, Richard M. ; WAGENAAR, Joris C.: Optimization methods for the Train Unit Shunting Problem. In: *Eur. J. Oper. Res.* 262 (2017), Nr. 3, S. 981–995. – URL https://doi.org/10.1016/j.ejor.2017.03.068

[13] HAALBOOM, Daniel ; BEŠINOVIĆ, Nikola: *Two-stage approach for rail freight terminal scheduling with mainline passenger traffic interaction*. 2024

[14] HAN, Peiran ; MENG, Lingyun ; LUAN, Xiaojie ; BEŠINOVIĆ, Nikola ; MIAO, Jianrui ; WANG, Yihui ; LIAO, Zhengwen: Integrated Optimization of Train Makeup Problem and Resource Scheduling in Railway Shunting Yards: A Hybrid MILP-CP Approach with Logic-Based Benders Decomposition. In: *Preprint submitted to Elsevier* (2024)

[15] HEIL, Julia ; HOFFMANN, Kirsten ; BUSCHER, Udo: Railway crew scheduling: Models, methods and applications. In: *European Journal of Operational Research* 283 (2020), Nr. 2, S. 405–425. – URL https://www.sciencedirect.com/science/article/pii/S0377221719304916. – ISSN 0377-2217

[16] KOVACS, Attila A. ; PARRAGH, Sophie N. ; DOERNER, Karl F. ; HARTL, Richard F.: Adaptive large neighborhood search for service technician routing and scheduling problems. In: *J. Sched.* 15 (2012), Nr. 5, S. 579–600. – URL https://doi.org/10.1007/s10951-011-0246-9

[17] LENTINK, Ramon: Algorithmic Decision Support for Shunt Planning. (2006)

[18] NES, Luuk van: Planning Drivers for Shunting Yards. In: *Universiteit Utrecht* (2023)

[19] PREIS, Henning ; POLLEHN, Tobias ; RUF, Moritz: Optimal resource rescheduling in classification yards considering flexible skill patterns. In: *J. Rail Transp. Plan. Manag.* 26 (2023), S. 100390. – URL https://doi.org/10.1016/j.jrtpm.2023.100390

[20] SALAZAR, José Arturo C. ; LANDA-SILVA, Dario ; QU, Rong: Workforce scheduling and routing problems: literature survey and computational study. In: *Ann. Oper. Res.* 239 (2016), Nr. 1, S. 39–67. – URL https://doi.org/10.1007/s10479-014-1687-2

[21] SPIEGEL.DE: *Bahn kündigt Ausbau der Direktverbindungen zwischen München und Paris an*. https://www.spiegel.de/wirtschaft/bahn-kuendigt-direktverbindungen-zwischen-muenchen-und-paris-an-a-ce42fe96-7418-41ef-bfd0-35cc00b1d3c2?sara_ref=re-so-app-sh. 2025. – accessed on 15 February 2025, 15:11

[22] WATTEL, Niek: Routing of shunt trains at logistics hubs of NS using Constraint Programming. (2021)