Universiteit Utrecht

Department of Information and Computing Sciences

Train Shunting and Service Scheduling: an integrated local search approach

Masters Thesis

Author:
Roel W. van den Broek
Utrecht University

First Supervisor:
dr. J.A. Hoogeveen

Second Supervisor:
dr. ir. J.M. van den Akker

Supervisor NedTrain:
ir. B. Huisman

August 2016

Abstract

Trains have to be maintained and cleaned regularly to ensure high passenger safety and satisfaction. These service tasks must be performed outside the rush hours, when the trains are parked off the main railway network at dedicated service sites. The activities on a service site are currently scheduled by hand; a difficult and time-consuming task that consists of matching incoming and outgoing trains, scheduling the service tasks, assigning trains to parking tracks and routing the trains over the service site. We propose a local search approach for the automated construction of such shunt plans that integrates these four planning aspects. Our heuristic is applied successfully to artificial and real-world planning problems, and outperforms a state-of-the-art mixed integer programming algorithm.

Preface

This thesis is submitted in partial fulfilment of the requirements for the degree of Master of Science in the Department of Information and Computing Sciences in the Graduate School of Natural Sciences. I was engaged in researching and writing this thesis from December 2015 to August 2016.

This project was performed at the request of NedTrain, where I undertook an internship for the entire duration of my research. From the start of my thesis, it was clear that the scheduling problem central to this study would be difficult to solve. Many years of research, including an PhD dissertation, have already been dedicated to this subject, yet the algorithms thus far were not competitive with human planners. Although this made the task at hand daunting at first, it also motivated me to explore different approaches to the problem, which ultimately resulted in a tool that successfully addresses the problem formulated by NedTrain.

However, this thesis would not have been possible without the cooperation of many others. First and foremost, I would like to express my gratitude towards my supervisors, Han Hoogeveen and Marjan van den Akker, for their excellent guidance and support in the past nine months. Their unwavering enthusiasm has supported me throughout my research and their constructive criticism allowed me to polish many parts of my thesis. Furthermore, I am grateful to my supervisor Bob Huisman and my colleague Demian de Ruijter at NedTrain for all the insightful discussions we have had on both the theoretical and practical aspects of service site scheduling. Especially at the start of my thesis this helped me tremendously to grasp the complexity of the problem. Last but not least, I would like to thank my parents for their moral support during the project.

I hope you will enjoy reading this thesis.

Roel van den Broek

August 2016

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this study we will focus on the service site scheduling problem in the Dutch railroad network. The service sites in the Netherlands are operated by NedTrain, a subsidiary of the Nederlandse Spoorwegen (Dutch Railways; NS), the largest Dutch railway operator. The NS transports more than 1.1 million passengers each day on one of the busiest railroad systems in the world. Within the NS, NedTrain is tasked with the maintenance, cleaning and refurbishment of rolling stock. To ensure that train passengers arrive both safely and comfortably at their destination, trains need to be maintained and cleaned at regular intervals. NedTrain has specialized maintenance depots, where most of the larger, technical maintenance and repair is done. More frequent tasks such as cleaning, washing, inspection and small maintenance are performed at service sites close to major stations, since these operations typically take no more than a few hours. An example of a service site can be seen in Figure 1.1. During the morning and evening rush most of the rolling stock will be used to transport the large number of commuters. In contrast, outside these peak hours, and especially at night, far fewer trains are needed to meet the demand, and the excess has to be parked away from the main railroad network. Part of the surplus is sent to the service sites for parking, and hence the sites function as shunting yards as well.

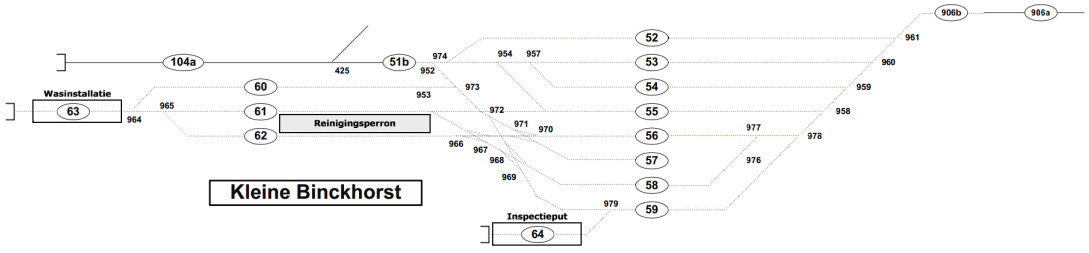


FIGURE 1.1: An example of a service site operated by NedTrain.

Plans for the service sites are created manually by the long-term planners at NedTrain. These shunt plans describe, for a twenty-four hour planning horizon, the assignment of incoming trains to departures the next morning, the order of completion of the service tasks, the tracks on which the trains will be parked and the exact route of each train movement. The arrival times are based on a fixed timetable, and service tasks are scheduled using their respective norm duration. Due to the proximity of the sites to major stations, most of which are located in urban areas, both the service and storage capacity is limited. Furthermore, train movements are highly constrained and trains should be parked such that each train movement has an unobstructed path to its destination. Consequently, the construction of conflict-free shunt plans is no trivial feat, and even for experienced planners a tedious and time-consuming process. Ideally, this task would at least be partially automated, producing plans that require only a few manual adjustments from the planners.

Although automated plan generation can be used as an operational decision-support system, the main motivation behind this study is to aid tactical and, to some extent, strategical decision making. Over the course of the next few years, NS will expand its fleet significantly to handle the ever increasing number of passengers. While additional rolling stock is beneficial to the everyday commuter, it causes more stress on the service sites. To evaluate whether the current facilities will be sufficient and, if not, where expansion is necessary, the capacity of the service sites has to be determined. Train carriages of older train types used by the NS have roughly the same size and NedTrain expressed the capacity traditionally as the number of carriages that could be parked simultaneously at a site. However, with the advent of new train types, the coach length is no longer uniform. Furthermore, the static parking measure provides no information on the actual processing capacity regarding the service tasks.

To obtain a more accurate capacity estimate, NedTrain is now defining the capacity of a service site as the maximum number of trains for which reliably a feasible shunt plan can be constructed. Due to the sheer number of scenarios that have to be evaluated to obtain an accurate estimate, it is not possible for the planners to create all shunt plans by hand, emphasizing the need for an automated plan generation system.

Unfortunately, previous attempts to automate the shunt planning either did not model all essential aspects of the service sites, such as the service task scheduling, or performed less than the planners at NedTrain. Over the past few year, the NS has developed a tool based on the state-of-the-art mathematical model formulated by Kroon et al.[21] to aid in the planning process at the service sites. However, this tool often fails to find feasible shunt plans, even for instances that are easily solved manually. To estimate the capacity of a service site accurately, the performance of any shunt plan generation tool

should at least be on par with human planners. Therefore, it is imperative to explore alternative approaches to the shunting problem.

The aim of this research is to develop an algorithm that can construct shunt plans for a twenty-four hour period, given the layout of the service site, a time table of arriving and departing trains and a list of service tasks that need to be completed. Both the arrival times and the service task durations are assumed to be deterministic.

Even without disturbances, finding a feasible shunt plan is a difficult problem, as it combines several well-known NP-hard problems. The service task scheduling can be viewed as a Resource-Constrained Project Scheduling Problem (RCPSP) with additional side constraints resulting from the parking. More specifically, it resembles an Open Shop Scheduling Problem with machine flexibility (multiple service facilities), buffer and blocking constraints (parking of trains), and release dates and deadlines (arrival and departure). To determine whether all trains can be parked, a Bin Packing Problem has to be solved. Furthermore, when trains have to be moved to allow another train to move over a track, the routing in the shunt plan strongly resembles sliding block puzzles such as the Rush Hour Problem. As challenging as these individual problems are, the algorithmic complexity of shunt planning arises mainly from the interaction of the matching, task scheduling, parking and routing components. The strong dependency between the different elements of a shunt plan makes it difficult to decompose the problem into multiple smaller, largely independent subproblems, a technique often proposed in literature for similar complex problems.

In the next chapter we will review research related to train scheduling problems in general and shunting problems in specific. An in-depth explanation of the problem central to this thesis is provided in Chapter 3. In Chapter 4 we introduce our local search model for the problem, and results for a number of real world instances will be presented in Chapter 5. Although the goal of this study is to construct shunt plans for the service sites operated by NedTrain, our algorithm can easily be applied to similar train (or tram) shunting and scheduling problems. To highlight the flexibility of our approach, multiple extensions and variants of the basic problem, as well as the necessary modifications to our local search model to cope with these modifications, are discussed in Chapter 6. Finally, the conclusions of this study and topics for future research are discussed in Chapter 7.

Chapter 2

Literature Review

In this chapter, we will provide a general overview of literature in the field of tram and train shunting problems, as well as an in-depth review of work related specifically to train shunting and maintenance. The research in this last section pertains mostly to shunting problems in the Dutch railway network.

2.1 General Shunting Problems

One of the earlier shunting problems discussed in literature is the problem of dispatching trams from a depot. Blasum et al. [2] have studied the assignment of trams stored in a depot with last-in-first-out tracks to round trips such that the number of shunt moves — moving a tram from one track to another — is minimized. They have shown that this problem is NP-hard and have proposed a dynamic programming algorithm to solve small instances of the problem. Winter and Zimmer [25] investigated the stored tram assignment problem extended with the assignment of arriving trams to tracks. Besides introducing an Integer Linear Program (ILP) to find the optimal solution, the authors focused on real-time decision-making to handle arrival delays of the trams. They have developed a number of heuristics that combine real-time information and the optimal solution computed with the ILP. These heuristics yield near-optimal results in less than three minutes for instances consisting of thirty to fifty trams.

The stored tram assignment problem introduced by Blasum et al. was shown by Eggermont et al. [10] NP-hard even if each track in the depot contains at most two trams. Furthermore, they have shown that the extended tram assignment problem described by Winter and Zimmer remains NP-hard when the instances are restricted to tracks that contain at most three trams

A survey of research on train sorting problems in shunting yards is provided by Gatto et al. [13]. In these problems, a single arriving freight train is split into individual cars. The cars have to be parked on some track such that they can depart in some predefined order. These freight cars have no engine and are parked by pushing them one by one over a hump. As the cars roll down the hump, they are guided through a tree of switches to arrive at their designated track. The problems are classified according to whether they allow extra shunt moves by locomotives and the objective function, such as minimizing the number of shunt moves or the number of tracks used. A broader overview of freight train sorting problems is presented by Boysen et al. [3]. Their survey includes practical aspects such as solution robustness and recovery, as well as research on train sorting problems with departure lateness minimization objectives.

The topic of robust train sorting is central to the work of Cicerone et al. [6]. They propose recovery rules that can be applied to their generated feasible solution to cope with disturbances in the input data, such as track unavailability or an unexpected car order in the incoming train. The authors have shown the trade-off between robustness and optimality, as well as the difficulty of creating solutions that are robust to multiple types of disturbances.

2.2 Passenger Train Shunting and Maintenance Problems

The train unit shunting problem (TUSP) was first introduced by Freling et al. [12] and consists of *matching* arriving train units to departing trains and *parking* these units on a track at a shunt yard. These train units are self-propelled and can be coupled to form a single, longer train. The authors use a decomposition approach in which a train unit matching is constructed first. In the matching problem, parts of the arriving trains are assigned to corresponding blocks of train units in departing trains. The objective of this subproblem is to find a matching that minimizes the number of times arriving trains have to be split to assign each train unit to exactly one position in a departing train. The corresponding mathematical model is solved using the standard MIP solver CPLEX. Secondly, the parts assigned in the matching problem are parked on a track at the shunt yard. A column generation approach, based on assignments of sets of train parts to each track, is used to find a feasible parking plan. To solve the pricing problem — construct a set of train parts that fits on the track such that each train part can leave on time without being blocked by another train — the authors propose a dynamic programming algorithm. The routing of the train parts on the shunt yard is not taken into account. They generated a shunt plan for a typical weekday at the shunt yard in Zwolle, consisting of eighty train units to be parked, in roughly half an hour.

Although Lentink et al. [23] use a decomposition approach similar to Freling et al. to solve the TUSP, they include the routes taken by the trains, and decompose the problem in four steps. First a matching is determined using the algorithm proposed by Freling et al. A graph representation of the shunt yard is presented in their study, which is used to estimate the routing costs from and to each shunt track. These estimates are used in the third step, the parking subproblem, to improve the column generation approach proposed by Freling et al. Finally, the actual routes are computed using the graph representation and the track occupation resulting from the previous step. Trains can move simultaneously and the entire path of a train movement is reserved for the duration of the move. The routes are computed sequentially and the order of evaluation is improved by a local search approach that swaps the order of two movements. The authors have shown that the time needed to generate a feasible shunt plan, including routing, for the shunt yard in Zwolle was around twenty minutes with their approach.

Instead of solving all components of the TUSP sequentially, Kroon et al. [21] construct solutions for the matching and parking subproblem simultaneously. This greatly increases the complexity of the problem, resulting in a mathematical formulation for the integrated approach that contains a large number of crossing constraints. Testing the model on a realistic case at the shunt yard in Zwolle revealed that there were over 400.000 constraints, which was too much for the CPLEX solver to find a feasible solution in a reasonable amount of time. To reduce the number of crossing constraints, the authors grouped them in clique constraints. This allowed them to find feasible solutions for their test case. Unfortunately, even with the reduction in constraints, the computation time increases rapidly with larger problems, taking several hours to complete.

An integrated approach has been investigated by van den Akker et al. [1] as well. They propose a greedy heuristic and an exact dynamic programming algorithm to the combined matching and parking problem. The heuristic uses track assignment and matching rules that select the locally best action on arrival and departure such that train units are parked in the correct order for the departing trains. The dynamic programming approach looks at all possible shunt track or matching assignments at each event on the shunt yard, and relies heavily on pruning nodes in the dynamic programming network that are unlikely to lead to the optimal solution as a way to reduce its computation time. In contrast to the model formulated by Kroon et al., both algorithms can include waiting time for the arriving and departing trains at the platforms. Furthermore, the exact algorithm is also capable of shunting a parked train unit to a different track, resulting in much more flexibility in the shunt plans. This property is difficult to include in the linear programming approaches proposed by other authors, due to the exponential increase in variables and constraints, even when allowing each parking interval to be split only once. The greedy heuristic is quite fast, but it is not capable of finding feasible solutions for

complex problems. Even with the pruning rules, the exact algorithm requires more than ten minutes to find a plan for a dozen train units, making it hard to use in practice.

In the work of Lentink [22] a practical extension to the TUSP is studied. Besides matching, parking and routing, the train units on a service site have to be *cleaned* as well. The cleaning subproblem is a crew scheduling problem, in which each train unit should be cleaned by a crew before it departs from the site. The first three steps are solved using the methods proposed in earlier work [12, 23]. The schedule for the cleaning crews is constructed last. The cleaning problem is modeled as a machine scheduling problem without preemption, where the machines correspond to the crews. The objective used for the machine scheduling problem is the minimization of the sum of the completion times. A mathematical model based on this formulation, in which the planning horizon is discretized into one minute blocks, is solved using CPLEX.

Jekkers [19] presented two genetic algorithms (GAs) for the integrated matching, parking and routing problem, both including waiting time at the arrival and departure platform. The algorithms have genes for the parking locations and the arrival and departure waiting times. One variant of the GA has also an extra gene for the matching, whereas the other one determines the matching using a greedy heuristic. The fitness of each member of the population is determined with a deterministic simulation. Routes are constructed during the simulation. This approach is applied successfully to generate shunting plans for shunt yards located near *Rotterdam Central Station* and *Hoofddorp*, two major stations operated by the NS group. The largest of the two instances, the shunt yard in Rotterdam, had seventy train units that needed to be parked, and took fifty minutes of computation time.

An integral approach is used by Jacobsen and Pisinger [18] to solve a train parking and maintenance problem. Each train has to be maintained at one facility or workshop located on the service site and parked before and after the service task. Using three metaheuristics, Guided Local Search, Guided Fast Local Search and Simulated Annealing, the authors attempt to construct schedules such that no trains are blocked by other trains, no departure delays occur and the makespan of the service tasks is minimized. Their results show that the local search approaches provide results close to shunt plans constructed by CPLEX, while taking only seconds of computation time compared to the twelve hours needed by the MIP solver. However, the largest instances contain no more than ten trains, with one maintenance task per train, which is not representative of real-world scenarios. Furthermore, the absence of routing and matching makes their approach not directly applicable to the scheduling problem for the service sites operated by NedTrain.

Van Dommelen [9] considered the train unit shunting problem with the extension of service tasks for a specific service site of NedTrain, the *Kleine Binckhorst*. Given a fixed matching, the goal is to generate feasible shunt plans that include servicing, parking and routing. The order of the service tasks are modeled mathematically as a flow shop problem, which is solved using CPLEX. The resulting parking intervals per train unit are then used as input for a tool called the OPG, which is developed internally by the NS, that determines both the parking locations and the routes. A feasible schedule for a test case with 35 train units was found after two hours of computation time. However, the OPG was not guaranteed to generate shunt plans without routing conflicts at that moment, suggesting that the reported number of parked train units might be an overestimate of the actual service site capacity.

Chapter 3

Problem Description

To ensure that all trains are clean and well-maintained at the start of their shift, a planner has to create shunt schedules that encompass all facets of the service site. Arriving trains should be assigned to departing trains. Furthermore, the planner has to decide on when the service tasks should be performed and, in case a service is offered on multiple tracks, where the tasks should take place. Whenever a train is not being serviced, it needs to be assigned to a track for parking. In the parking track assignment, the planner has to take into account both the length of the track and the order of parking. The latter is to avoid conflicts when trains depart from the track. Furthermore, the routes driven by the trains from track to track have to be specified in the shunt plan.

While some elements of a shunt plan are difficult to properly schedule by themselves, especially when the number of trains and tasks is large, the main complexity of the shunting problem follows from the interaction between the different components. Adjustments to the matching or the service schedule result in different parking intervals, and whether there is an unobstructed path from one track to another depends entirely on the current track occupation of the service site.

In this chapter, we will provide a glossary of the terms used to describe the problem, followed by an overview of the Train Unit Shunting Problem (TUSP) as described by Lentink [22]. We will illustrate the problem examined in this study with an example at the end of the chapter.

3.1 Preliminaries

The infrastructure of a service site consists of a set $\mathcal{T}$ of *tracks* which are connected by a number of *switches*. Figure 3.1 shows a small example service site. A track $\tau \in \mathcal{T}$

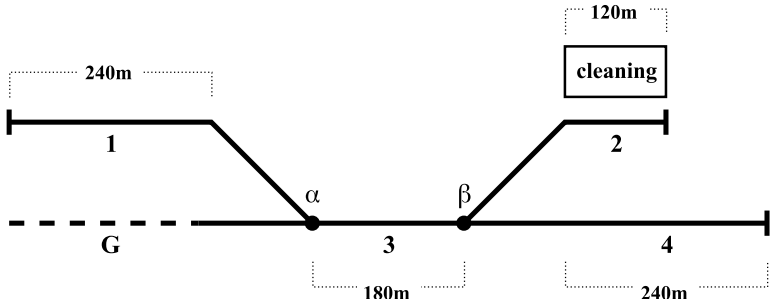


FIGURE 3.1: An example of a service site. Trains enter and exit the site over gateway-track G and can be parked on tracks 1 to 4. The tracks are connected by the switches α and β. The length of the parking tracks is displayed in meters. A cleaning platform allows internal cleaning tasks to be performed on trains positioned on track 2.

has a *length* of l_τ and two end-points which are referred to as the *A*- and *B-side* of τ. Tracks approachable from both sides are called *free* tracks and their behavior regarding the parking of trains is best modeled as a deque (doubly-ended queue). *LIFO*-tracks are only accessible from one side, with the other side being blocked by a bumper, and are, as their name suggest, similar to a stack. Examples of both track types can be seen in Figure 3.2. Define $\mathcal{T}_{free}$ to be the set of all free tracks and $\mathcal{T}_{LIFO}$ the set of LIFO-tracks, then $\mathcal{T}_{free} \cup \mathcal{T}_{LIFO} = \mathcal{T}$. A service site is connected to the rest of the railway network through one or more tracks known as *gateway*-tracks.

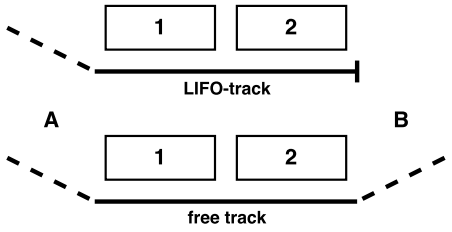


FIGURE 3.2: A LIFO-track (top) and a free track (bottom), depicted by the solid horizontal lines, each with parked trains 1 and 2. Access to the tracks is shown by the dashed lines. The LIFO-track is only approachable from the A-side (left), and has a bumper on the B-side. As train 2 was first to enter the LIFO-track, it cannot depart before train 1. In contrast, train 2 can leave the free track without conflicts, albeit only from the B-side.

Each *service task* $\sigma \in \mathcal{S}$ that must be performed on the service site has a duration of d_σ and a set of tracks $\mathcal{T}_\sigma$ on which the task can take place. The service tasks can be divided in two categories:

- The *track-specific* tasks $\mathcal{S}_{spec}$ require a certain *facility* that is located along a track, such as the train wash or a cleaning platform (see Figure 3.1). Only a single train can be processed at a time in each facility.
- Contrarily, multiple *track-independent* tasks $\mathcal{S}_{indep}$ can be done simultaneously on each possible track, although a *service crew* is needed to perform each of the tasks.

The facilities and crews are collectively referred to as *resources* in this study.

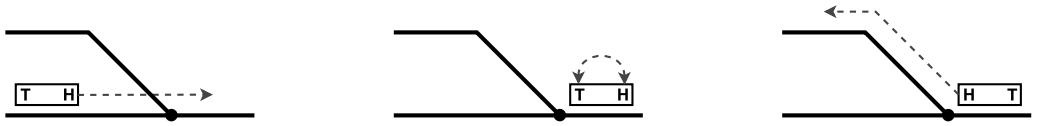
FIGURE 3.3: Two train units of the same train type, ICM, but different subtypes: the top train unit consists of three carriages (ICM-3); the bottom one has four (ICM-4).

The trains we consider in this study consist of *train units*, bi-directional and self-propelling vehicles that move without a dedicated locomotive. All train units on a service site are referred to as the set TU. Train units $tu \in TU$ are classified according to *types* (t_{tu}) and *subtypes* (st_{tu}). An example of two train units is shown in Figure 3.3. Two train units can only be combined to one longer train if their train types are equal. The subtype indicates the number of carriages of a train unit, and therefore determines its length. The set of all service tasks belonging to the same train unit tu is denoted as $\mathcal{S}_{tu}$.

On a service site the incoming trains can be split and combined to form new train compositions. These temporary compositions are referred to as *shunt trains*. During its stay at a service site, a train unit is always part of a shunt train, although the specific shunt train can vary over time. Likewise, a train unit cannot be part of two shunt trains simultaneously, since at each split or combine one or more new shunt trains are formed and the old shunt trains cease to exist. Each shunt train can have a number of *predecessors* and *successors*, which describe the transition of train units between different shunt trains due to splitting and combining.

In the example in Figure 3.1, a train, entering the service site through the gateway-track, might be planned to park on track 1. The train can only move to track 1 by reversing on one of the tracks 2 to 4. This reversal of direction is called a *saw move*. To accomplish this manoeuvre, the control of the train has to be transferred to the tail of the train, which then becomes the new head of the train. Furthermore, the driver must walk to the other end of the train to resume the movement, as only a single driver is available per train. In general, the saw moves are time-consuming operations, and hence avoided by human planners if possible. The steps involved in a saw move are illustrated in Figure 3.4

A *crossing* is defined as the (undesired) scenario in which the path of a moving train is blocked by one or more other trains. For example, if train 2 in Figure 3.2 attempts to leave the LIFO-track before train 1, its path is obstructed by train 1 and a crossing will occur. A shunt plan that contains a crossing is infeasible.



(a) Move to the reversal track (b) Transfer control (c) Move to the destination

FIGURE 3.4: The steps of a saw move to get the train, depicted as a rectangle, from its original track to the top track. It moves to a track on which the saw manoeuvre is possible (a), before transferring control from one end to the other (b). As a result the original head (H) of the train becomes its tail (T) and vice versa. The train can then move to its destination (c).

3.2 Train Unit Shunting Problem

Lentink identified four subproblems in the Train Unit Shunting Problem, namely *matching*, *parking*, *routing* and *cleaning*. For the purpose of this study we will expand the notion of cleaning to service tasks in general. In the following subsections we will describe the different components of the TUSP in detail and indicate any differences in our problem formulation and the problem as described by Lentink [22].

3.2.1 Matching

In the matching problem, we are given a set of arriving trains AT and a set of departing trains DT. Each arriving train $at \in AT$ has an arrival time a_{at} and consists of a sequence of one or more train units $(tu_1^{at}, \cdots, tu_m^{at})$. No two arriving trains have a train unit in common and the set of all train units in the arriving trains equals the set TU. Similarly, a departing train $dt \in DT$ departs at time d_{dt} and specifies a sequence of one or more train subtypes $(st_1^{dt}, \cdots, st_n^{dt})$. The arrivals and departures are mixed over time, i.e. a departure can happen before all trains have arrived.

The objective in the matching problem is to match each train unit tu in an arriving train at to a specific position i in one of the departing trains dt such that

1. $st_{tu} = st_i^{dt}$, the train subtypes match, and
2. $a_{at} \leq d_{dt}$, the train unit enters the service site before departure.

In general, we can refine the second condition to include the duration of the service tasks of train unit tu,

$$a_{at} + \sum_{\sigma \in \mathcal{S}_{tu}} d_\sigma \leq d_{dt}, \tag{3.1}$$

since the train unit has to spend at least that amount of time on the service site. Note that this is a necessary, yet not sufficient condition for the trains to depart on time. Other factors such as the duration of the train movements and the schedule of the service task can still cause delayed departures.

When two train units, coupled at arrival, are assigned either to two different departing trains or in a different order to a single departing train, they have to be split on the service site. Similarly, if train units assigned to the same departed train are not coupled (correctly) at arrival, a combine operation has to take place. For most trains, each splitting operation takes two minutes and a combine three minutes. Therefore, it is usually preferable to find a matching that minimizes the number of changes to the train compositions.

In practice, parts of the matching are already predefined, that is, some train units are already assigned to a specific position of a departing train. These train units are scheduled in advance to ensure that they reach one of the maintenance depots in time for larger maintenance. The fixed matches simplify the matching problem significantly, although it limits the flexibility at the service site by forcing undesirable splits and combines.

We assume the service site to be empty at both the start and the end of the day. Trains parked for multiple days are easily modeled by adding dummy arrivals before all other arrivals and corresponding dummy departures after the other trains have left the service site.

It was shown by Lentink [22] that the problem of finding a train unit matching that minimizes the number of splits and combines, without even taking the routing and service task durations into account, is NP-hard in the strong sense by a reduction from the 3 PARTITION problem.

3.2.2 Task Scheduling

For each train unit, we are given a set of service tasks together with their respective durations. Each service task requires some resource for its entire duration and the availability of resources is usually limited. Furthermore, we assume no precedence relations between service tasks, neither between tasks of different train units nor among tasks belonging to the same unit. In the task scheduling problem, a schedule must be constructed such that all tasks are completed and all trains can depart on time.

Although services are specified for individual train units, in practice the tasks are carried out on the shunt train level, meaning that all service tasks of the same resource are

performed as one for a shunt train, with the corresponding duration equal to the sum of the separate train unit tasks. As a result, the actual tasks that have to be scheduled depend on the composition of the shunt trains.

If we assume all arriving and departing trains to consist of a single train unit, the service task scheduling simplifies to a generalization of the *Open Shop Scheduling Problem* (OSSP), a well-known NP-complete problem. Specifically, we get an open shop scheduling problem with machine flexibility, release dates and deadlines. In this formulation, the different resources are the machines. Each train unit is a job, with its service tasks corresponding to operations in the OSSP. The release dates and deadlines of an operation depend on the arrival and departure of the train unit.

3.2.3 Parking

A shunt train that moves to a track τ will be added in front of one of the two sides of the sequence of trains already parked on the track. This side is fixed if τ is a LIFO-track, otherwise it depends on the arrival side of the train. The main objective in the parking problem is to place all shunt trains on the tracks such that at any moment in time the combined train length does not exceed the length of the track, while simultaneously ensuring that each shunt train has an unobstructed path off the track when it has to depart. The order of trains on a free track depends not only on the order of arrival, but also on the arrival side. Therefore, the arrival side has to be specified as well when we park a train on a free track.

An additional constraint is that parking, splitting and combining is not allowed at the gateway-tracks, which in particular means that all shunt trains of a departure composition should be parked on one track in the correct order to combine the trains. To avoid conflicts in the shunt plan, it is allowed to park a train temporarily at a different track, a property that is exploited in the work of van den Akker et al. [1].

For the case that the parking intervals of all shunt trains are known, the problem turns out to be NP-complete. This was proven by Lentink [22] using a reduction from the BIN PACKING problem.

3.2.4 Routing

The aim of the routing problem is to find for each train movement an unobstructed path that minimizes the movement duration. Many functions, with varying levels of detail, can be used to approximate the duration of a route. For the service sites operated

by NedTrain, we estimate the duration based on the number of tracks, switches and required saw moves on the path, using the following values.

- Traversing a track takes sixty seconds, regardless of train type or length.
- Transitioning from one track to another requires thirty seconds for each switch along the way. This is a rather crude estimate, as switches can be operated either automatically or by hand, depending on the service site. For automatic switches, the thirty seconds is a realistic duration, whereas roughly two minutes are needed if the driver has to operate the switch by hand. However, the manual switches will be on average half the time already in the correct position, requiring no operating time. Furthermore, it is usually possible for one of the staff to operate the switch in advance, which justifies the thirty seconds approximation. The duration is independent of the train composition.
- In contrast, the time it takes to perform a saw move does depend on both the train type and the train length. The reversal of a train requires that the control of the train is transferred to the other side and the driver must walk from one end to the other. The process of reversing a shunt train usually takes somewhere between two and four minutes, while the time it takes for the driver to walk to the other side of the train is estimated to be between twenty to thirty seconds per carriage. Note that when a train performs a saw move, it traverses the track used for the reversal twice, which has to be taken into account during the computation of the duration of a route.

These durations have been validated by the planners at NedTrain to provide a good approximation of the actual movement durations observed on the service site.

In contrast to the work of Lentink, we do not allow simultaneous movements on the service site. Due to safety regulations, train controllers, of whom there is often only one present at a site, may only guide a single train movement at any time. Although these regulations are in reality often violated due to delays, an initial shunt plan should follow the correct procedure, thus significantly simplifying the routing problem. As routes can no longer intersect, there is no longer any interaction between them, meaning that the computation of one route is independent of other routes. Furthermore, we can define a linear ordering on all movements on a service site.

It can occasionally happen that while a train is moving, another train arrives according to the timetable. However, since gateway-tracks are not available for parking, the incoming train has to move immediately to a different track, thus violating the single movement

rule. Therefore, in order for a plan to be feasible, it should ensure that no other train moves during an arrival.

We can find a shortest path (or determine that no path exists) in polynomial time if we assume the track occupation on the site at the time of the movement to be fixed. However, we are allowed to move parked trains out of the way and position them elsewhere. The problem of deciding whether there exists a sequence of train movements such that a shunt train reaches its destination closely resembles the decision variant of the sliding block puzzle RUSH HOUR, a problem known to be PSPACE-complete [11].

3.3 Example

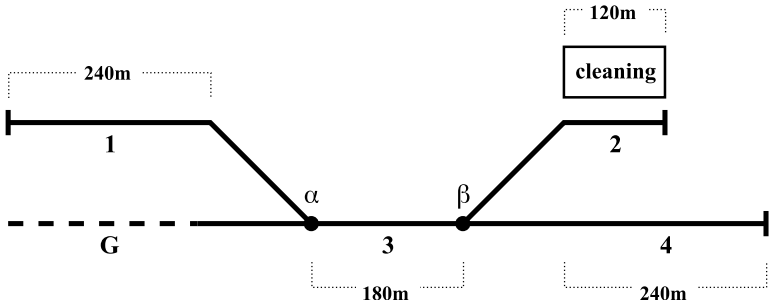


FIGURE 3.1: An example of a service site. Trains enter and exit the site over gateway-track G and can be parked on tracks 1 to 4. The tracks are connected by the switches α and β. The length of the parking tracks is displayed in meters. A cleaning platform allows internal cleaning tasks to be performed on trains positioned on track 2.

To illustrate the train unit shunting problem, let us consider a simple scenario of three train units at the service site in Figure 3.1. There are two arriving and two departing trains in this example, which are scheduled according to the timetable in Table 3.2. The train units are of the ICM type, depicted in Figure 3.3. The duration of a saw move of an ICM train is four minutes plus an additional thirty seconds per carriage. Two train units are scheduled for an internal cleaning, as can be seen in Table 3.1.

Train Units	Type	Service Tasks
1	ICM-3 (82m)	cleaning (34 minutes)
2	ICM-3 (82m)	cleaning (34 minutes)
3	ICM-4 (107m)	none

TABLE 3.1: The train units in the example scenario.

No predefined matching is provided in this scenario, and hence we can freely assign the train units 1 and 2 to either the first departure or the right-most position in the second departing train. The first arriving train has to be split, because there is no match for the entire train and the train is longer than the length of track 2, which provides access

Arriving Train	Time
(1, 2)	12:00
(3)	12:30

Departing Train	Time
(ICM-3)	12:45
(ICM-4, ICM-3)	14:00

TABLE 3.2: The arrivals and departures in the example scenario. The departure trains specify the composition of subtypes instead of the train units, since the assignment is part of the matching problem. The ordering of the train units or subtypes indicates from left to right the order of the train units or subtypes in the train on the service site.

Train Unit	Consecutive Locations (Activity)
1	G (A) → 3 (P) → 4 (P) → 2 (C) → 1 (P) → G (D)
2	G (A) → 3 (P) → 2 (C) → G (D)
3	G (A) → 1 (P) → G (D)

TABLE 3.3: The locations of each train unit in chronological order. The activity at each location — arrival (A), parking (P), cleaning (C) and departure (D) — is displayed between parentheses.

to the cleaning platform. Furthermore, there will be one combine operation to form the composition of the second departure. With only a single cleaning platform, we have to decide in which order the service tasks will be completed. Additionally, we have to park the three train units when they are not being cleaned in such a way that the trains can depart on time without any crossings.

Let us start the construction of a feasible shunt plan by matching incoming to outgoing trains. We assign train unit 2 to the train departing at 13:00; the other two train units will be part of the second departure. As train unit 2 is the first to depart, we schedule it to be cleaned before train unit 1. A feasible assignment of parking locations is shown in Table 3.3. The corresponding movements and composition changes are listed in chronological order in Table 3.4.

The movement durations are based on the values provided in Section 3.2.4. For example, the movement of the last departure travels over six tracks (a reversal track is counted twice) and crosses each switch on the service site twice, which requires $6 \cdot 1 + 4 \cdot {}^1\!/_2 = 8$ minutes. The saw move adds an extra $4 + 7 \cdot {}^1\!/_2 = 7{}^1\!/_2$ minutes based on its length, for a total of $15{}^1\!/_2$ minutes. Note that this shunt train has to reverse on track 4, as track 3 is too short for this manoeuvre.

This example illustrates the main complexity of the Train Unit Shunting Problem. Although the individual shunting subproblems — matching, servicing, parking and routing — are seemingly trivial to solve, the interaction between these components will make most shunt plans infeasible. Although parking on track 3 is possible, it blocks virtually all routes on the service site. Furthermore, poorly parked trains might require multiple saw moves, which can easily cause departures to be delayed in this heavily time-constrained example. The service task schedule is determined entirely by the matching,

Start Time	End Time	Shunt Train	Activity	Track(s)
12:00:00	12:02:30	(1,2)	Movement	G → 3
12:02:30	12:04:30	(1,2) → (1), (2)	Split	3
12:04:30	12:07:00	(2)	Movement	3 → 2
12:07:00	12:09:30	(1)	Movement	3 → 4
12:30:00	12:41:00	(3)	Movement	G → 3 → 1
12:41:00	12:45:00	(2)	Movement	2 → 3 → G
12:45:00	12:55:30	(1)	Movement	4 → 3 → 2
13:29:30	13:33:30	(1)	Movement	2 → 3 → 1
13:33:30	13:36:30	(3), (1) → (3, 1)	Combine	1
13:44:30	14:00:00	(3, 1)	Movement	1 → 3 → 4 → 3 → G

TABLE 3.4: The movements and composition changes of trains in a shunt plan for the example scenario provided in Tables 3.1 and 3.2.

as there is not enough time to clean both train units of the first arriving train before one of them has to depart. The matching is dependent on the parking and routing as well; switching the order in which train units 1 and 2 depart will result in an infeasible solution due to the small time-window between the first arrival and departure.

Other examples of the interactions between the different components of the TUSP are listed below.

- As trains might be forbidden to park on tracks that are part of some facility, service tasks can be scheduled such that fewer trains have to be parked simultaneously.
- Some tracks might only be reachable by moving through some facility, which creates a dependency between the routing and servicing subproblems.
- As mentioned in Subsection 3.2.1, whether a matching is possible depends both on the service task schedule and the routes that are taken. The latter is in turn mostly determined by the location of the parked trains.

Chapter 4

A TUSP Local Search Model

In the past few years, the NS has developed a tool for the matching, parking and routing subproblems of the TUSP. This tool, called the OPG, first estimates the routing cost of shunt from one track to another, similar to the approach taken by Lentink et al. [23]. Secondly, the matching and parking subproblems are solved simultaneously, using the cost estimates of the routes to find track assignments that simplify the subsequent routing problem. To find a matching and a parking assignment, a problem formulation based on the mathematical model introduced by Kroon et al. [21] is solved by CPLEX. In the final step of the OPG, many routes are generated for the movements, and CPLEX is used to find a feasible combination of these routes. The OPG allows movements to be performed simultaneously. To determine the capacity of the service sites, the OPG has recently been extended to include the scheduling of service tasks. This extension greedily schedules tasks in order of the earliest possible starting time such that no pair of tasks of the same train unit are executed simultaneously. The movements required by the resulting schedule are then supplied to the other components of the OPG. The task scheduler determines the tracks for the track-specific tasks, while the locations of track-independent tasks are assigned by the matching and parking module of the OPG. Although the OPG is able to construct shunt plans for small instances, the results are currently unsatisfactory in more realistic cases. The OPG is capable of planning fewer train units than human planners, due to its limited flexibility — most notably, it lacks the option to shunt a train to a different track during a parking interval — and excessive computation time for larger instances.

Since the mathematical models developed thus far have proven to be either too complex or too restrictive to solve the TUSP in a reasonable amount of time, it is worthwhile to explore different techniques for solving the train unit shunting problem. Heuristics based on decomposing a problem into smaller, independent parts are often applied to

similar problems, because the individual subproblems are usually much simpler to solve than the parent problem. However, a decomposition method is unlikely to be effective for the TUSP due to the complex interactions between the different components. Therefore, we propose an integrated approach based on local search to the train unit shunting problem in this study. Local search algorithms gradually improve some candidate solution, a shunt plan in case of the TUSP, by making small changes, and have been applied numerous times in the field of Operations Research with great success. A stochastic variant of local search, Simulated Annealing [20], is used in this study, but many aspects of our proposed method apply to similar heuristics such as Tabu Search [14, 15] as well.

In this chapter we will present all the components of our local search approach to the Train Unit Shunting Problem. We start with an overview of the simulated annealing technique, followed by the objective function used in our algorithm. A graph representation of shunt plans is introduced next, which, together with a set of local search operators, can be embedded within the simulated annealing framework to construct good solutions for the TUSP. We conclude this chapter with a procedure to find an initial shunt plan for the local search.

4.1 Introduction to Simulated Annealing

Local search is a heuristic method that attempts to find the *global optimum*, a solution that minimizes some *objective function*. Starting from an initial solution, local search algorithms move through the search space, the set of all solutions, by iteratively applying small, local changes. The procedure of making a certain type of adjustment to a solution is referred to as a *search operator*. The *neighborhood* of a solution is the set of all solutions reachable from the current solution through a single application of a search operator.

A local search algorithm has to decide which solution is selected from the neighborhood as a candidate solution for the next iteration of the search process. This does not necessarily have to be the best neighbor, or even a neighbor solution with a better objective value than the current solution. In the naive local search approach called *hill-climbing* we repeatedly move to a better neighbor until we reach a position in the search space where all neighbors are worse. This position is referred to as a *local optimum*. Although the global optimum is a local optimum as well, search spaces often contain many local optima, and the objective value of a local optimum does not have to be close to the global optimum. Since there are no better neighbors around a local optimum, the hill-climbing approach cannot escape from a potentially bad solution. Therefore, more

sophisticated local search algorithms have mechanisms to move away from local optima, accepting a worse solution in hopes of finding a better area in the search space later on.

Much research has been devoted to creating better search strategies. One extensively studied technique is *Simulated Annealing* [5, 20], a stochastic search process that has seen many successful applications to other computational problems. A simulated annealing algorithm randomly selects a neighbor and accepts it immediately as the candidate solution for the next iteration if it is an improvement over the current solution. If the selected solution is worse, it is selected with a certain probability depending on the difference in solution quality and the state of the search process. Let b be the selected neighbor of the current solution a, and suppose we are minimizing an objective function f. If $f(b) > f(a)$, then the probability of acceptance p is

$$p = e^{\frac{f(a)-f(b)}{T}}, \tag{4.1}$$

where T is a control parameter called the *temperature*. Every Q iterations, we multiply T with a value $0 < \alpha < 1$ to gradually decrease the probability of accepting a deterioration over the course of the search. The algorithm ends when a stopping criterion, such as a maximum running time or time without improvements, is fulfilled and returns the best solution found during the entire search.

4.2 Objective Function

A solution to the train unit shunting problem is a shunt plan that consists of

1. an assignment of incoming train units to positions in the departing trains, as well as the splits and combines required for the matching,
2. a service task schedule, containing for each service task a starting time and a resource,
3. the parking track (and arrival side) for each time period that a train is idle, and
4. the order of movements along with the routes taken by each.

The aim of our local search algorithm is to find a shunt plan that respects all constraints described in Chapter 3. Our heuristic is built around the concept of resource feasibility. We ensure in all generated plans that each service resource, whether facility or crew, has at most a single task at any time. No simultaneous movements are allowed as well, since the drivers can be viewed as a type of resource. Additionally, a proper

matching of arriving and departing trains is ensured during the entire algorithm, i.e. no type mismatch can occur. The other constraints in TUSP, most notably regarding the departure time and the routing conflicts, are added as soft constraints in the objective function.

We can compute the following properties for a shunt plan p.

- $\mathrm{cr}_p(m)$ = the number of crossings of movement m.
- $\mathrm{tlv}_p(\tau)$ = the number of occasions in the entire shunt plan that the combined train length of shunt trains occupying track τ exceeds the length l_τ.
- $\mathrm{delay}_p(d)$ = the delay of departure train d.
- $\mathrm{delay}_p(a)$ = the delay of arrival train a. Note that an arrival delay can only occur when a train arrives during the movement of another train. Trains are not allowed to stand still on a gateway-track.
- $\mathrm{gc}_p(d) = 1$ if departure train d requires a combine operation on its gateway-track, and 0 otherwise.

A shunt plan is feasible only if none of these values are greater than zero. During the local search, we have to determine whether a neighbor solution is preferable over the current shunt plan, i.e., estimate if it is "closer" to being feasible. To the end, we assign penalties to the undesirable properties in the objective function and search for a solution that minimizes the costs. The cost of shunt plan p is computed with the equation

$$\begin{aligned}
\mathrm{cost}(p) \;=\; & w_{cr} \cdot \sum_{\text{movement } m \in p} \mathrm{cr}_p(m) & + \\
& w_{tlv} \cdot \sum_{\tau \in \mathcal{T}} \mathrm{tlv}_p(\tau) & + \\
& w_d \cdot \sum_{\text{departure } d \in p} \mathbb{1}(\mathrm{delay}_p(d)) \quad + w'_d \cdot \sum_{\text{departure } d \in p} \mathrm{delay}_p(d) & + \\
& w_a \cdot \sum_{\text{arrival } a \in p} \mathbb{1}(\mathrm{delay}_p(a)) \quad + w'_a \cdot \sum_{\text{arrival } a \in p} \mathrm{delay}_p(a) & + \\
& w_{gc} \cdot \sum_{\text{departure } d \in p} \mathrm{gc}_p(d) & + \\
& w_m \cdot |\{m \mid \text{movement } m \in p\}| \,,
\end{aligned} \tag{4.2}$$

where $\mathbb{1}$ is the indicator function. In this equation, each property of the shunt plan is multiplied with a corresponding weight w. Although it is sufficient to include only the existence of delays in the objective function, as even a delay of one second is still not acceptable, the total delay is used to guide the local search. This is based on

the assumption that smaller delays are usually more easily resolved. The number of movements is penalized, as a shunt plan with minimal moves is preferred by the planners at NedTrain. Although the movement count is not used to determine the feasibility of a shunt plan, it can aid the local search by directing it to more promising areas in the search space.

4.3 Shunt Train Activity Graph

Essential to any local search algorithm is a solution representation that properly captures all important aspects of the solution, while simultaneously allowing for easy modification through the local search operators. This is especially important for the Train Unit Scheduling Problem with its tightly connected supproblems.

To represent the different characteristics of a shunt plan and ensure that all hard constraints are respected, we model each activity in the shunt plan as a node in a precedence graph. The resulting *Shunt Train Activity Graph* — referred to as *STAG* or simply *Activity Graph* in the remainder — is a directed acyclic graph, where the arcs indicate the precedence relations between the nodes. The activity graph for the example shunt plan constructed in Section 3.3 is shown in Figure 4.1. We distinguish multiple node types based on their activity, each type having its own set of data associated to the node. Our graph model consists of the following node types.

1. The *arrival* nodes contain information on the arrival time and track.
2. Similarly, a *departure* node stores the departure time and track.
3. The *service* nodes keep track of their service tasks, such as cleaning, and assigned resource.
4. *parking* nodes.
5. A *movement* node stores the entire path from origin to destination together with its duration.
6. The *departure movement* node is a specialized movement node that always precedes a departure node. If not all shunt trains are combined before departure, the departure movement node contains multiple routes to the departure track; one for each shunt train that is not yet combined.

Additionally, all node types except the movement node store information on the assigned track and arrival side. The first four node types are referred to as *track activity nodes*. The arcs in the graph express precedence constraints of

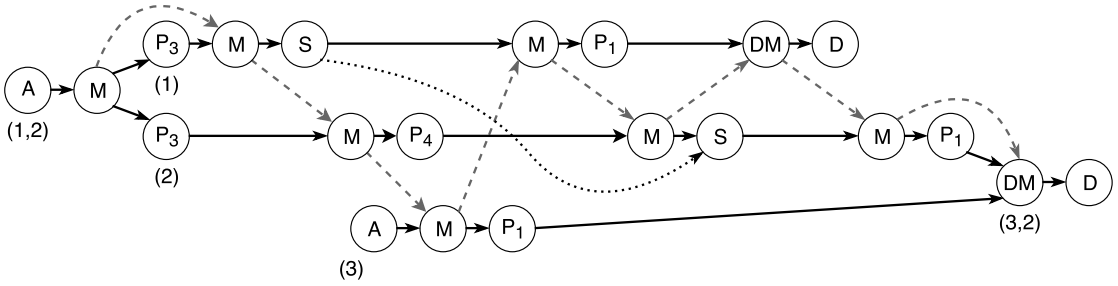


FIGURE 4.1: The activity graph of the shunt plan in Section 3.3. The arrival (A), service (S), parking (P), movement (M), departure movement (DM) and departure (D) nodes are connected by arcs indicating the precedence relations. Solid, black arcs represent the order of operations of one or more shunt trains. The corresponding train units of the nodes are between parentheses. The grey, dashed arcs determine the order of the movements, and the black, dotted arc indicates which service task is completed first. The assigned track for each parking node is shown in subscript. Other information stored at the nodes is omitted for clarity.

1. activities either of the same *shunt train* or between a predecessor-successor-pair of shunt trains,

2. *movements* or

3. *service* tasks assigned to the same resource.

All nodes, except for the movement nodes, have at most one incoming and one outgoing arc, both connected to a movement node. Movement nodes have at least one incoming and one outgoing arc, with more than one incoming arcs indicating a combine operation and multiple outgoing arcs when splitting. Here, we assume that splits always happen directly after arriving on a track and combines just before departure from a track. Examples of splitting and combining can be seen in Figure 4.1 at the first movement node respectively the departure movement nodes. If, for example, a shunt train stays on the same track for parking after a service operation, there still is a movement node separating the two activities in the STAG, albeit one with an empty route and no duration.

The movement and service orders are modeled explicitly in the activity graph. Furthermore, the matching follows directly from the shunt train compositions at the departure nodes. The parking and route information is stored at the corresponding nodes; hence the corresponding shunt plan can easily be inferred from the activity graph.

Due to the strong dependencies between the different elements of the TUSP, few adjustments to a shunt plan are truly local. Even a simple change of track of a parking node in the STAG can affect all movement during the entire parking interval. Therefore, whenever a local search operator modifies the current STAG, we not only need to determine

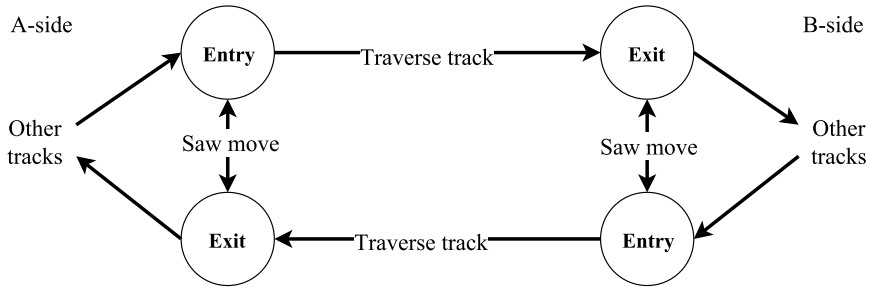


FIGURE 4.2: A single track at the service site is split into four vertices, two at each side. Arcs model the possible moves of a train on the track and the connection to other tracks on the service site.

the cost of the new solution, but also evaluate the routes and finish times of multiple nodes. In the next two sections we will discuss how these components are updated.

4.4 Routing

After each alteration, we recompute all routes that were potentially affected by the change. To find the shortest path for each movement, a model of the railroad infrastructure at the service site is needed. Similar to the approach taken by Lentink [22], we model the infrastructure as a graph with sets of four vertices for each track. Each track has two vertices on either side, one for entering and one fore exiting the track on that side. The arcs in the graph represent the possible manoeuvres of a shunt train. Arcs connecting the entry vertex on one side to the exit vertex on the opposite side allow trains to move over the track, while arcs between the two vertices on the same side indicate a saw move on that particular side of the track. Access to the track is modeled through incoming arcs at the entry vertex and outgoing arcs from the exit vertex to other tracks.

The objective in the routing problem is to find the path with the shortest durations among the routes of minimal crossing count. We can model this problem as a single-source shortest path problem by assigning weights to each arc in the graph. These weights consist of two components, one based on the duration of the action corresponding to that arc and one indicating the cost in terms of crossings. We can map the durations described in Section 3.2.4 directly to weights for the different arc types.

- The arcs that connect the two sides of a track have a weight of sixty seconds.
- Access arcs, used to move from one track to another, have a penalty of thirty seconds for each switch between the two tracks.

- The time it takes to perform a saw move, and hence the weight of the corresponding arcs, depends on the composition of the train. The weight of a saw move arc is for each train composition simply the time in seconds needed for the manoeuvre.

The weights that penalize paths with crossings depend on both the train length and the occupation of the tracks during the movement. Moving over a track results in a crossing whenever there are one or more trains positioned on the track. During a saw move, a crossing occurs if the total amount of free space on the track is less than the length of the reversing shunt train. We count the situation where the track is empty, yet the train is longer than the track, as a crossing as well, since this is not allowed in a feasible shunt plan. As shunt trains are not allowed to park on the railroad that connects the individual shunting tracks, they can always move along those arcs in the graph without the risk of a crossing. The actual weight of a crossing is chosen such that it dominates the duration weights of all possible routes to prefer paths with fewer crossings over a shorter, but more conflicted path.

By expressing the routing problem as a single-source shortest path problem, we can apply standard search algorithms such as Dijkstra's algorithm [8] to find the least conflicted route with the shortest duration for each movement.

The frequent shortest path computations are expected to have a significant impact on the running time of the entire local search algorithm, as each adjustment of the activity graph likely requires several movements to be recomputed. Therefore, we propose two methods to improve the speed of solving the routing subproblem. Firstly, we note that the number of train compositions is fairly limited in practice. Each train type only has a few subtypes and the maximum length of a train is restricted, as it has to fit on some station platform. This allows us to compute for each train composition the path with the shortest duration between every pair of tracks on an empty service site in advance. The solutions to these all-pairs shortest path problem can then be used as a heuristic to guide the search for the shortest path in case the service site is not empty, extending Dijkstra's algorithm to the informed search algorithm $A*$ [16]. Both Dijkstra's algorithm and A* search for the shortest path from source s to target t by constructing partial paths — the shortest path to a set of discovered but not yet expanded vertices — and repeatedly expanding the partial path that is estimated to be the shortest path to t. Let, for any two vertices u, v, $d(u, v)$ be the distance from u to v including crossings and let $h(u, v)$ be the distance without. Dijkstra's algorithm expands the partial path ending at v that minimizes $d(s, v)$, whereas A* uses $d(s, v) + h(v, t)$ as estimate for the expansion order. For A* to find the shortest path, the heuristic has to be *admissible*, meaning that it never overestimates the distance of one track to another. The admissible criterion is clearly satisfied by the shortest duration heuristic, as it provides a perfect estimate if

the shortest path is unobstructed and underestimates the length of the path whenever there is a crossing. The heuristic allows us to find the shortest path more quickly than with a plain Dijkstra's algorithm, although its effect becomes less evident as the number of occupied tracks grows.

Secondly, the crossings depend on the occupation of the tracks, or the state, of the service site at the time of the movement. It is likely that, during the computation of the shortest path, we encounter a state similar or identical to some previous state. This can be exploited by storing each computed route along with the current state and using a cached route if the origin and destination are the same and the states are similar. To determine whether two states are similar, we only consider the potential crossings along the route. This allows us to express a state as a set of Boolean values, two per track, that indicate the existence of a conflict when reversing on or crossing the track. Constructing this expression can easily be done if the current state of the track and the shunt train of the movement are known. Note, however, that no information on the shunt train is stored, hence in similar states we store the same route for all shunt trains. This route is not necessarily the optimal path for all shunt trains, as the time it takes to perform a saw move depends on the train type and the number of carriages. In practice the duration of saw moves dominates the time required to move over the tracks and switches on the route, thus resulting in very few suboptimal paths. Furthermore, the most important aspect, the number of conflicts, will be the same in both the stored and the shortest path, suggesting that this improvement is a favorable trade-off. To prevent the cache mechanism from consuming too much memory, it is advisable to limit its size and remove old entries once the cache starts to fill.

4.5 Time

Once the routes have been computed, the arrival and departure punctuality can be computed. We evaluate the movements in order and determine the earliest start time of a movement node as the maximum end time over all predecessor nodes in the activity graph. The end time is then the start time plus the duration of routing and, if applicable, splitting or combining.

The departure movement nodes are a special case, since a departing train needs to arrive exactly on time at the gateway track, instead of when it is finished with its tasks. For these nodes, we set the start time as close as possible to the scheduled departure time minus the duration, without preceding any predecessor nodes.

As the time computation requires only constant time per node in the activity graph, its effect on the overall computation time is likely to be dominated by the routing subproblem.

4.6 Search Neighborhoods

In the next four subsections, we will outline the neighborhoods that will be explored for new solutions by our algorithm. The neighborhoods are grouped by the subproblems — routing, parking, servicing or matching — affected by their corresponding local search operators.

Our local search approach does not include a neighborhood for the splitting and combining of trains. Although the activity graph is capable of modeling any configuration of splits and combines of the shunt trains, we split each arriving train at most once, at the first track the train is moved to after arrival. Similarly, each shunt train is combined no more than once, just before departure from the service site. Furthermore, an arriving train is split precisely into the parts determined by the current matching and the length restrictions based on the service tasks, as described in Subsection 3.2.1. The matching itself can be modified by applying a local search operator, and any change in matching is likely to result in different splits and combines.

This heuristic approach to splitting and combining removes the dimensions of how and when to change shunt train composition from the search space of our local search algorithm, reducing its size considerably. This allows us to focus solely on the remaining components of the shunt planning process. Regarding the performance impact on our algorithm of the splitting and combining restriction, it is unlikely that all feasible shunt plans are outside the reduced search space. On one hand, splitting and combining only once per train reduces the time spent on composition changes. On the other hand, the early splits and late combines mean that the shunt trains, and thus the service task durations, are short for most of their stay at the service site, which in general simplifies the service scheduling and parking subproblems. These two changes are expected to provide a good balance between the routing durations, which include splitting and combining, and the service and parking scheduling, that favour short trains and services; hence an operator that changes train compositions would only be usefull in rare circumstances.

4.6.1 Routing

The routes taken by the shunt trains are recomputed whenever the track occupation changes, and as such, no local search operator is needed for the path-finding component

of the routing problem. However, we can attempt to improve a shunt plan by reordering the movements. Suppose that train a is parked on a LIFO-track. If a train b arrives on the same track just before train a departs, a crossing will occur. In this case, it is beneficial to let a depart before b arrives. The search neighborhood of rearranging movements is denoted as the *movement shift* neighborhood. The corresponding local search operation consists of selecting a movement node and shifting it earlier or later in the linear ordering imposed on the train movements. To ensure that the resulting shunt plan is valid, a number of constraints have to be respected. Shifting node n to an earlier or later position, before respectively after node m, is only allowed if

1. the nodes n and m have no train unit in common and
2. no service tasks assigned to the same resource overlap.

If shifting a node to an earlier or later position of the linear ordering of movements is not allowed, then the same holds for all further positions in the respective direction.

In some cases, we can improve the shunt plan by reducing the number of movements. Consider the scenario where we have shifted a node next to a movement node of the same shunt train. Suppose that the node between the two movements in the activity graph is a parking node. Then the shunt train has to move from some activity to the track corresponding to the parking node, wait there some time and then continue to its next activity, whereas in the meantime, no other changes in activity happen on the service site according to the shunt plan. When such a scenario occurs, we can simply skip the parking activity and merge the two movement nodes.

Merging the movement nodes means that there is no buffer time between two other activities. This can severely limit the flexibility of subsequent local search operators. Therefore, all operators can insert an extra parking activity into the graph whenever buffer time between two activities is necessary.

4.6.2 Parking

Conflicts in the shunt plan such as crossings and trains exceeding the track length can often be solved through changes in the parking activities. We propose three different search neighborhoods for our local search approach that modify the parking location of shunt trains.

1. The *parking reassignment* neighborhood simply consists of all shunt plans that can be constructed by changing the track and arrival side of one parking activity. This

means that for each free track on which we can park the shunt train, there will be a solution for both arrival at the A-side and the B-side in the neighborhood. If multiple parking activities have the same movement node as predecessor in the activity graph due to a split, we change the assignment for all involved parking nodes. Otherwise, a single train movement would end at different tracks, and the shunt plan would be invalid.

2. Similarly, the *parking swap* neighborhood is the set of solutions obtained by swapping the track and arrival side of two parking activities that overlap in time.

3. To overcome one of the drawbacks of the OPG, we introduce the *parking reposition* neighborhood. The corresponding local search operator splits a parking node into two, with a movement node in-between. Either the first or the second parking node is assigned to a different track, the other one remains the same. This allows us to move a train that is blocking the path out of the way or position a train such that it can be combined before departure. Due to the number of parking activities, possible track assignments and positions for the new movement node in the linear ordering of movements, we only apply this operator when it can potentially solve a conflict. This means that we limit the operator to cases where a conflict, such as a crossing, occurs on the same track as the parking activity and position the new movement node resulting from the split just before the movement leading to the conflict.

   One of the main reasons to include the number of movements in the objective functions is to prevent this local search operator from introducing an excessive number of extra parking activities.

4.6.3 Servicing

The local search operators that adjust the resource assignment and order of the service tasks are based on operators proposed in the literature on similar problems such as the Job Shop Problem [7], the Open Shop Problem [24] and their counterparts with machine flexibility [4].

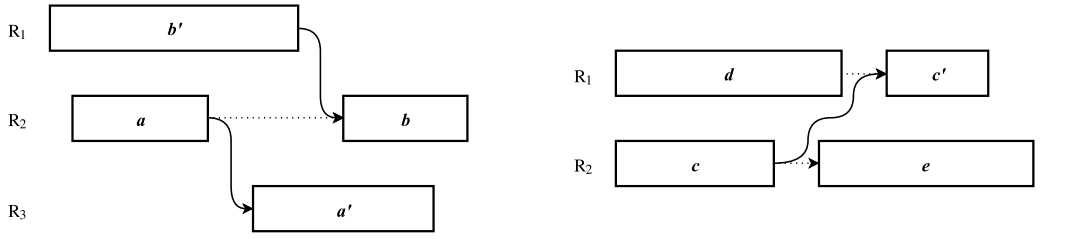
1. All valid solutions that can be constructed by swapping the order of two consecutive service tasks assigned to the same resource are part of the *resource order swap* neighborhood.

2. Similarly, the *train order swap* neighborhood contains the shunt plans in which the execution order of two consecutive service tasks of the same shunt train are swapped.

3. In a solution in the *resource reassignment* neighborhood, a single service task is assigned to a valid position in the task schedule of a different suitable resource. The order of tasks of the shunt train remains the same.

The order of the movements to and from the rescheduled service tasks often has to be updated as well. These movement nodes are inserted in the ordering on the movement activities such that tasks using the same resource do not overlap and the time between the movements to and from the same task is close to the duration of the task.

As with the movement shift neighborhood, we have to take care that we preserve the acyclic property of the activity graph of all neighbors, because activity graphs with cycles correspond to invalid solutions. For the resource and train order operators, generating invalid solutions can easily be avoided by checking whether the start of a subsequent task of the first shunt train respectively resource occurs before the end of a previous task of the second shunt train / resource. If this is the case, the swap is not valid, as can be seen in Figure 4.3. The resource reassignment operator can only assign a service task to a resource if the task schedule of the resource has a position available between predecessor and successor tasks of the service task.



(a) Swapping tasks a and b assigned to resource R_2 is not possible

(b) Swapping tasks c and c' belonging to the same shunt train is not possible

FIGURE 4.3: Infeasible service task schedule swaps in the *resource order swap* (a) and the *train order swap* (b) neighborhoods. The swaps are not possible, as both require the order of other tasks to be adjusted as well. Dashed arcs indicate resource order; solid arcs represent precedence relations of tasks of the same shunt train.

4.6.4 Matching

Changes to the matching are made by interchanging the assignment to positions in departing trains of two (partial) shunt trains. The matching of these parts can only be swapped if their composition is identical, none of the train units has a fixed match and, for both parts, the arrival time plus the sum of the service tasks durations is less than the assigned departure time. The *train unit swap* operator corresponding to these interchanges can cause the composition of the shunt trains to change. Train units might

become part of a different shunt train, multiple shunt trains can be merged or a single shunt train is split into parts. We update the activity graph by connecting the last service or parking node of each shunt train to the correct departure movement node. Cycles in the graph can occur if a movement of the newly assigned train units is scheduled later than the departure movement; hence we adjust the position of the departure movement in the movement ordering if necessary to avoid invalid shunt plans. When there is a departure delay, we move the departure movement to an earlier time if possible.

Since service tasks are executed on the shunt train level, the change in train composition can result in new tasks that have to be assigned to resources and ordered. If the new service tasks are the result of a split, they can often simply be inserted back into the task schedule at the position of the service task they were originally part of. When multiple service tasks are merged, we first attempt to insert the new task at the position of one of the original tasks. If no allowed position is found, we look for any available resource that has a position available that causes no resource conflicts, allowing the tasks of the shunt train to be reordered. Otherwise, we randomly select one of the positions of the original tasks and delay all other subsequent tasks assigned to the corresponding resource.

Furthermore, each of the parking activities of the new shunt trains have to be assigned to a track. When splitting, we simply use the track assignment of the original shunt train. For merged shunt trains, we select for each parking activity the track that results in the smallest number of conflicts. The track occupation over time is often changed significantly by the swaps, thus requiring many train movements to be reevaluated.

The train unit swap operator is highly disruptive to the shunt plan, with far less similarity between neighboring solutions constructed by this operator than neighbors in the search neighborhoods described in the previous sections. Instead of operating locally, the solutions in this swap neighbor more or less resemble restarts of the local search algorithm. Therefore, we propose another, more restricted neighborhood to use in conjunction with the train unit swap operator. This *shunt train swap* neighborhood is a subset of the train unit swap neighborhood, in which we only consider swapping the assignments of entire shunt trains and do not merge shunt trains. In these solutions, no shunt train will be split and individual train units cannot be assigned to a different shunt train. Although the corresponding local search operator is far less flexible than the train unit swap operator, it has the property that the shunt train composition, and thereby the set of service tasks, is preserved between neighbors. This limits the changes in the activity graph to the departure movement nodes. Ideally, we would only use the train unit swap operator if major modifications to the matching have to be made, and search for solutions in the shunt train swap neighborhood otherwise.

4.7 Initial Solution

In the previous sections we have described our local search algorithm for the train unit shunting problem. However, every local search requires an initial solution to start with. One of the properties of simulated annealing is the tendency to move away quickly from the initial position in the search space, as it often accepts a deterioration in solution quality at the start of the search. As a result, any decent shunt plan suffices as initial solution. On the other hand, the shunt plan has to adhere to the hard constraints posed in Section 4.2. Therefore, we will present a simple sequential algorithm to construct the first shunt plan.

We start with the matching subproblem. A perfect matching between the incoming and outgoing train units is constructed such that no arriving unit is matched to a position on a train that departs before all tasks of the unit can be finished. This constraint on the matching is similar to the necessary condition for entire shunt trains in Equation (3.1). The arriving units and departing types form a bipartite graph, which allows us to construct the matching using the well-known matching algorithm of Hopcroft-Karp [17]. Note that we can immediately abort the search for a feasible shunt plan if no perfect matching can be found.

From the train unit matching we can derive the minimum number of splits and combines that have to be performed to transform the incoming trains into the desired departure compositions. Train units coupled on arrival can only remain together if

1. all are assigned in the same order to consecutive positions on a departing train,
2. their arrival time plus the sum of the duration of their service tasks is less than the departure time, and
3. for each service task there is a track adjacent to the required facility that is long enough to harbor all train units at once.

Although we have the *train unit swap* operator to change the composition of the shunt trains, it is desirable to start with a matching that has few splits and combines. To reduce the number of composition changes, we use a simple simulated annealing algorithm which uses only one local search operator that swaps the assignment of train units with equal train subtypes, regardless of time constraints. The objective function we try to minimize consists of the number of splits and combines and a penalty for each train unit that cannot depart on time. This local search runs for a fixed amount of time, no more than a few seconds, and returns a feasible matching that minimizes the cost function.

A task schedule is constructed next. The tasks of each shunt train and their due date — the departure — result from the matching in the previous step. In order of increasing due date, the tasks are added one by one to the schedule of a corresponding resource. If a task can be assigned to multiple resources, the resource with the currently smallest total processing time is selected. The order of tasks of the same shunt train is determined by the earliest starting time of the tasks in their resource schedule. Ties in the shunt train task order are broken randomly.

Movements from and to each service task are assumed in the initial heuristic, that is, each shunt train will be parked between each service task. Furthermore, there is a movement for both arrival at and departure from the service site. For each task, we can determine its start and end time using the earliest starting time rule, i.e. scheduling the tasks as early as possible. These times are then used to define a linear ordering on the movements, breaking ties between simultaneous movements at random.

Finally, we need to decide where each shunt train will be parked between arrival, service tasks and departure. The track for each parking interval is picked randomly from the tracks with sufficient free space during the entire interval, or assigned randomly if no such track is available.

Chapter 5

Results

In this chapter, we compare the performance of the simulated annealing approach (SA) proposed in Chapter 4 with the OPG tool developed by NS on both artificial and real-world test cases. The test cases are based on one of the service sites operated by Nedtrain, the "*Kleine Binckhorst*", which is situated near The Hague Central Station. The Kleine Binckhorst is a service site of medium size, and consists mostly of free tracks. An overview of this service site is provided in Figure 5.1. The gateway-tracks that provide access to the Kleine Binckhorst are track 906a and 104a. Parking and reversing on these tracks is not allowed. There are three dedicated service facilities: a washing machine (*"wasinstallatie"*) on track 63, a platform for internal cleaning (*"reinigingsperron"*) between tracks 61 and 62 and an inspection pit (*"inspectieput"*) at track 64. Only a single train can be cleaned externally at the washing machine. There are two crews at the cleaning platform, allowing a train to be cleaned at each track adjacent to the platform. Since the pit is used only for unplanned maintenance, service tasks related to the pit are not part of the shunt plan. However, trains must not be parked at track 64. The track-independent maintenance checks that are carried out by service crews at Kleine Binckhorst can take place on any track that is not part of some facility. The saw move duration and the average service task duration used in the test cases are listed in tables 5.1 respectively 5.2.

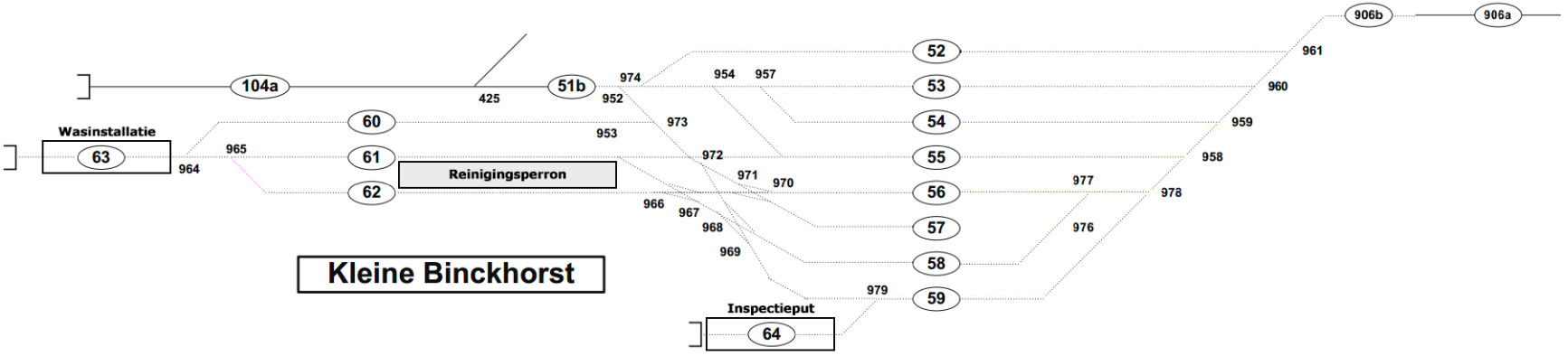


FIGURE 5.1: Service site "*Kleine Binckhorst*".

Train Type	Saw Move Base	Saw Move Carriage
SLT	2	$^1/_3$
VIRM	4	$^1/_2$
DDZ	4	$^1/_2$

TABLE 5.1: The saw move duration of the train types in minutes. The duration consists of a base time required to transfer control and additional walking time per carriage.

Train Type	Train Subtype	Length	Internal Cleaning	Washing	Maintenance Check
SLT	SLT-4	70	15	23	23
SLT	SLT-6	101	20	24	27
VIRM	VIRM-4	109	37	24	11
VIRM	VIRM-6	162	56	26	14
DDZ	DDZ-6	154	56	26	18

TABLE 5.2: The train length in meters and the average service task duration in minutes for each train subtype.

The main goal of this study is to determine the capacity of the service site. As the cost of expanding a service site is far greater than that of hiring extra personnel, we can assume for this particular problem that there are always sufficient service crews available on site. As a result, we can simplify our simulated annealing approach by removing the track-independent service tasks from the task scheduling problem. Although the track-independent tasks still have to be performed, we can easily check for each of those tasks whether the corresponding shunt train is parked at a suitable track long enough to complete the task. Tasks that remain unplanned can either be penalized in the objective function or inserted back into the activity graph as service nodes to ensure that they will be planned in later iterations of the local search. The settings used by both the simulated annealing algorithms with and without explicit maintenance checks can be found in Table 5.3. In the next section we will evaluate the effect of this improvement by comparing the results of the simulated annealing approach that penalizes unplanned tasks (SA w.o. checks) with the original algorithm proposed in Chapter 4 (SA).

Setting	T	α	Q	w_{cr}	w_{tlv}	w_d	$w_{d'}$	w_a	$w_{a'}$	w_{gc}	w_m	w_{ut}
Value	8	0.97	2000	10	15	40	0.05	20	0.03	20	0.02	10

TABLE 5.3: The settings of our local search approach used in the tests. The simulated annealing settings are discussed in Section 4.1. The weights w of the objective function are described in Section 4.2. The penalty for unplanned track-independent service tasks is denoted by w_{ut}.

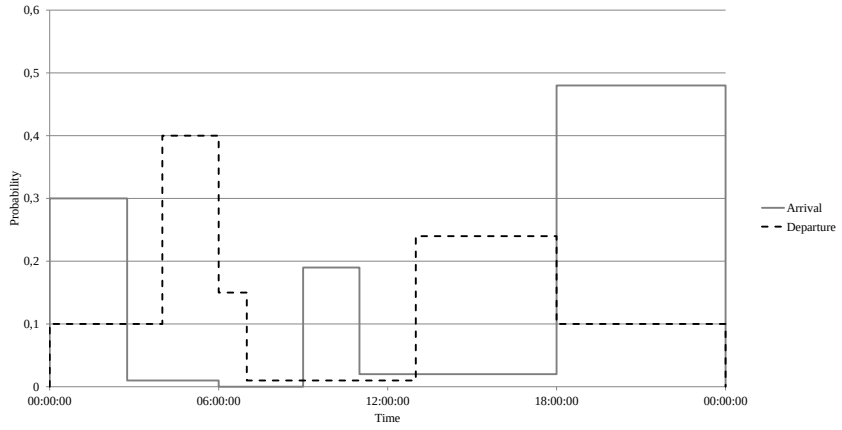


FIGURE 5.2: The PDFs of the arrival and departure times of trains in the artificial test cases.

Train Subtype	Arrival	Internal Cleaning	Washing	Maintenance Check
SLT-4	0.28	1.00	0.16	1.0
SLT-6	0.17	1.00	0.16	1.0
VIRM-4	0.41	1.00	0.16	0.58
VIRM-6	0.10	1.00	0.16	0.58
DDZ-6	0.04	1.00	0.16	0.58

TABLE 5.4: The Arrival column shows the distribution of the subtypes over the arriving trains. The probability that a task has to be performed on a certain train unit is shown in the last three columns.

5.1 Artificial Scenarios

To evaluate the performance of the different algorithms, we generated test cases based on distributions of time tables and service tasks that resemble a normal weekday at the Kleine Binckhorst. Figure 5.2 shows the probability density function of the arrival and departure times. Note that the departure times will be slightly biased, since we sample the distribution again if a train would depart before its arrival. The train type and service task distributions are shown in Table 5.4. The scenarios do not have a fixed matching; the arrival-departure assignment has to be performed entirely by the tested algorithms. The maximum length of composite trains in our test cases is three train units, and approximately half the arriving and departing trains is composed of two or more train units.

Our test cases have a varying number of train units, and for each number of train units we generated ten scenarios. The number of feasible plans found by the two simulated annealing algorithms (SA and SA w.o. checks) and the OPG are shown in Figure 5.3.

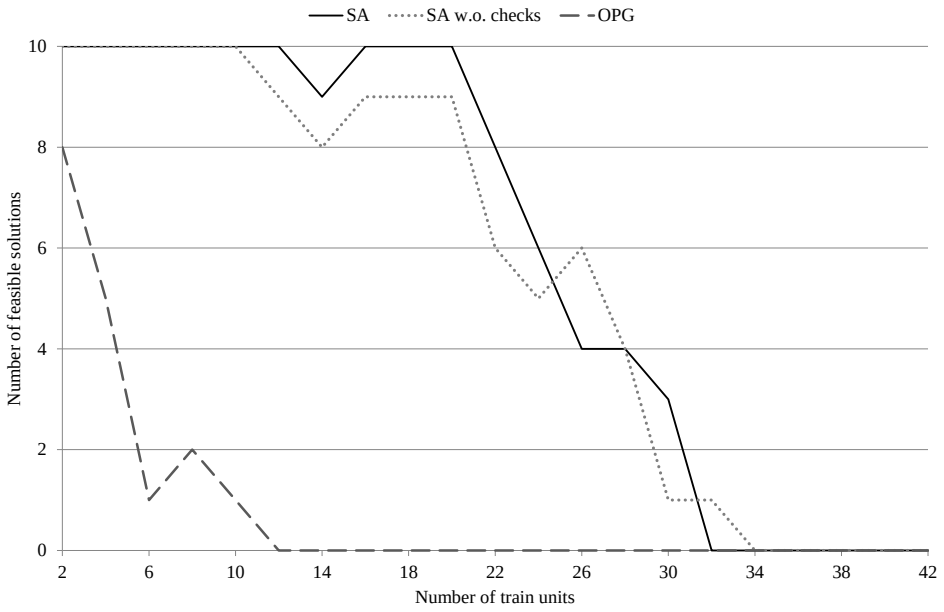


FIGURE 5.3: The number of feasible shunt plans found in ten runs for each the artificial test cases with service tasks. The results of three algorithms are shown: the simulated annealing heuristic *(SA)* described in Chapter 4; the modified simulated annealing approach without explicit maintenance checks proposed at the start of this chapter *(SA w.o. checks)*; and the *OPG*.

A comparison of the running times of successful runs can be seen in Figure 5.4. The algorithms stop when a feasible solution is found. A limit of five minutes of computation time is given to all algorithms.

The tests show that the local search algorithms are capable of planning more train units than the OPG. As we can see in Figure 5.3, our heuristics manage to find feasible shunt plans in half of the cases with 24 train units, whereas the OPG fails to generate solutions for scenarios with more than ten train units. More importantly, the performance of the OPG is much less reliable. Even with as few as four train units, it struggles to find feasible shunt plans consistently. In contrast, the algorithms proposed in this study continue to perform well up to twenty trains. Although there are only a few data points due to the limited number of instances solved successfully by the OPG, Figure 5.4 suggests that the average running time of the OPG already starts to increases rapidly with small instances.

The two simulated annealing approaches show a more graduate rise in computation time, both requiring less than 90 seconds for 24 train units. Interestingly, the two local search algorithms perform quite similar. The running times as well as the number of feasible

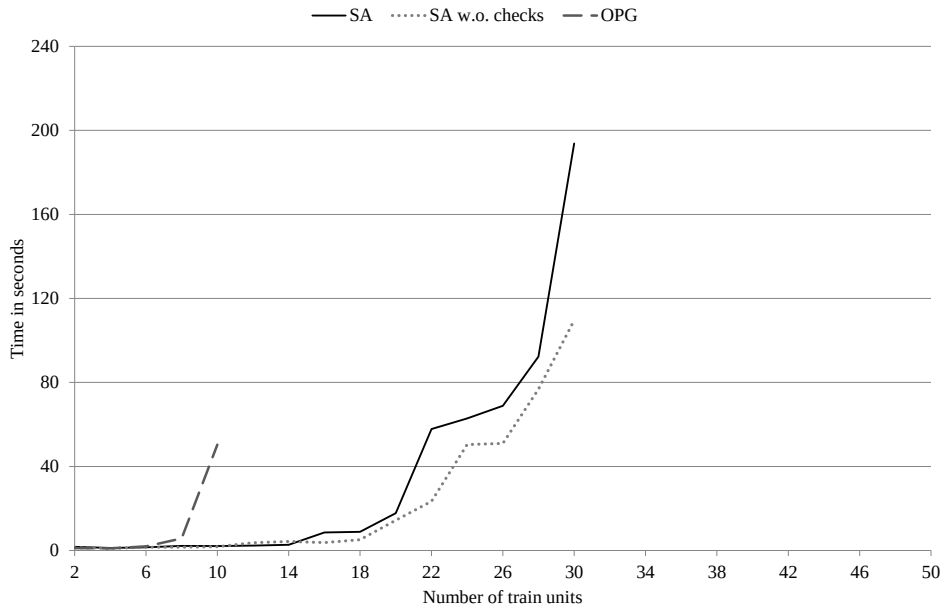


FIGURE 5.4: The average running time in seconds needed by the three algorithms to generated the feasible shunt plans depicted in Figure 5.3 is plotted against the number of train units in the test cases.

solutions found are largely comparable, suggesting that planning the maintenance checks has little impact on the construction of the overall shunt plan. This is consistent with the planning process of the planners at NedTrain, as they often insert these short and flexible maintenance checks in the service schedule after determining all other aspects of the shunt plan.

The comparison between the simulated annealing approach and the OPG is not entirely fair. The matching, parking and routing modules of the OPG are the result of years of research, whereas the priority rule for the service scheduling is an ad hoc solution to be able to answer the capacity question posed by NedTrain. To eliminate the effect of the task scheduling module in the OPG, we compare the performances of the algorithms on test cases without service tasks, again varying the number of train units and generating the scenarios based on the distribution in Figure 5.2 and Table 5.4. The results of the conducted experiments are visualized in Figures 5.5 and 5.6. Note that the simulated annealing approaches with and without explicit maintenance checks will perform identically in the absence of service tasks; hence we only report their results once. Instead, we ran additional tests of our simulated annealing algorithm, this time without the *parking reposition* neighborhood (SA w.o. parking reposition operator). Removing this search operator brings the parking flexibility of our approach closer to that of the OPG, which

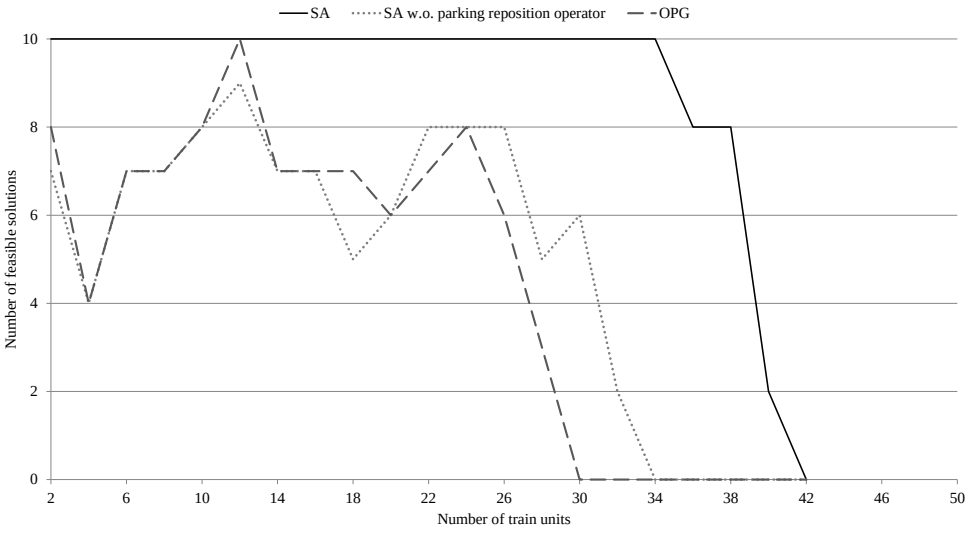


FIGURE 5.5: The number of feasible shunt plans found in ten runs for each the artificial test cases without service tasks. The results of three algorithms are shown: the simulated annealing heuristic *(SA)* described in Chapter 4; the simulated annealing approach without the parking reposition operator *(SA w.o. parking reposition)*; and the *OPG*.

allows us to investigate the impact of this search space reduction on the ability to find feasible shunt plans.

As can be seen in Figure 5.5, the number of trains that the service site can handle increases remarkably when no tasks have to be performed. This suggests that the current bottleneck of the site is most likely one of the service facilities. The performance of the OPG is closer to our algorithms when no service tasks have to be scheduled. Furthermore, its reliability has been improved significantly, implying that the OPG would benefit greatly from a better task scheduler. Nevertheless, our heuristic is still capable of generating feasible shunt plans for higher numbers of train units, and manages to do so in less time than the OPG.

Comparing the results of the tests of the simulated annealing approach with and without the parking reposition operator shows that the increased parking flexibility is needed when the service site contains many train units. The heuristic without parking repositioning performs close to the OPG, indicating that increasing the parking flexibility of the OPG might be necessary to produce better results. However, even without the parking reposition neighborhood the local search approach performs slightly better than the OPG, a surprising result given the fact that the OPG allows simultaneous movements on the service site.

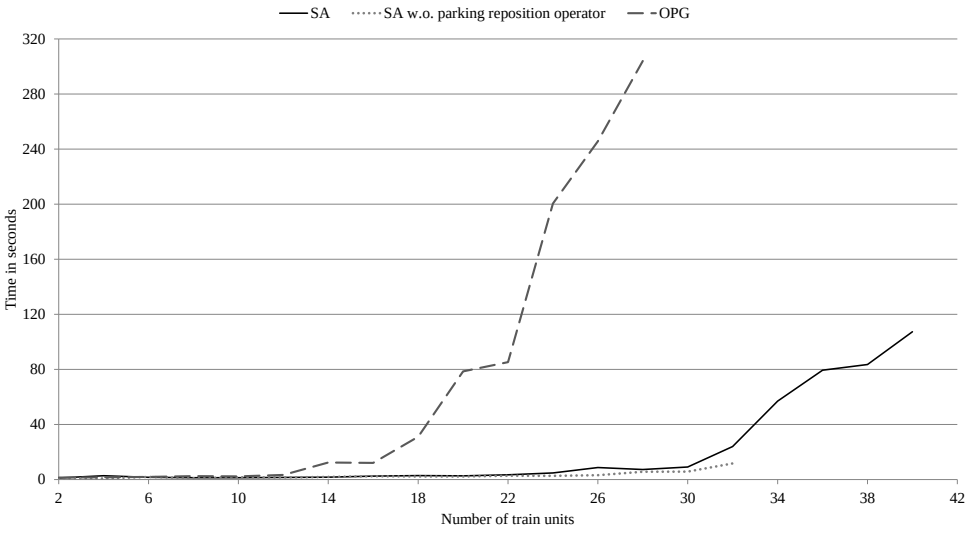


FIGURE 5.6: The average running time in seconds needed by the three algorithms to generated the feasible shunt plans depicted in Figure 5.5 is plotted against the number of train units in the test cases.

An interesting result is the poor performance of both the simulated annealing approach with restricted parking flexibility and the OPG on instances with less than twelve train units. This might be explained by the limited number of possible matchings. When it is not possible to shunt a parked train to a different track, even simple instances with only two train units can be infeasible. For example, if an arriving train composed of two train units of different subtypes has to depart with the order of the train units reversed, then it is not possible to solve this instance. Both splitting and combining cannot happen on gateway-tracks, and thus at least one of the two train units has to be moved from their parked location to a different track in this case. If sufficient train units are available on the service site, we can simply change the matching to avoid this situation. Therefore, smaller instances are more likely to be impossible to solve for the two algorithms. Similarly, instances where the matching is already (partially) fixed will be harder when parking flexibility is limited.

Figure 5.6 shows the same image as Figure 5.4 regarding the running times of the algorithms. Most notably, the OPG requires several minutes even with few train units. The difference in computation time between the two simulated annealing variants is likely caused by the reduced number of neighbors when the parking operator is not applied.

Note that, although the Kleine Binckhorst is just one of many service sites operated by NedTrain, its characteristics, such as the service types and the number of facilities, are

shared with most other sites. Furthermore, no planning strategies developed specifically for the Kleine Binckhorst are included in our local search approach, suggesting that our algorithm will be able to handle similar scenarios for other service sites as well.

5.2 Real-world Scenario

In order to be able to support human planners in their decision-making process, our algorithm should be capable of solving the real-world instances that are currently planned manually by the long-term planners at NedTrain. Therefore, we have tested our local search approach on a scenario of a normal week day at the Kleine Binckhorst as well. This instance consists of 32 train units, arriving and departing in 23 respectively 21 trains. Due to the time table, the maximum number of train units simultaneously on the service site is 25; these train units occupy 77 per cent of the total track length available for parking. There are 59 service tasks that must be completed: 27 internal cleanings, 25 maintenance checks and 7 train washes. The distributions of the train types and the time table correspond to the data in Figure 5.2 and Table 5.4.

We have used the simulated annealing approach described in Chapter 4 to search for a feasible shunt plan for the test case, which was found after four minutes of computation time. In this solution, the 23 arriving trains are split into 27 shunt trains. The shunt plan contains 88 shunt movements, of which 32 contain a saw move. The large number of saw moves result in an average movement duration of 10 minutes, meaning that almost 15 hours of train movements are needed in this 24-hour shunt plan. In 14 cases a parked train is shunted to a different track to make room for another train.

This test case shows that our local search algorithm can be used as a decision-support system for NedTrains planners. Although the planners indicated that the algorithm is a useful tool to generate base plans which they can further expand themselves, they would like to see more details specific to the individual service sites included in the shunt plans. In the next chapter, we will investigate extensions to our heuristic to include more aspects of the planning such as planner preferences and additional constraints on the service site.

Chapter 6

Service Site Model Extensions

Although the formulation of the train unit shunting problem approximates the components of service site scheduling well enough to determine the capacity, it does not capture all practical aspects that a human planner has to take into account to construct a shunt plan. Additionally, regulations can differ between service sites and tend to change every few years. Therefore, it is vital that a shunt plan generation tool can be adapted to handle new, practical requirements without much effort. In this chapter we will examine the implications of several extensions of and modifications to the basic TUSP model on our algorithm. Adjustments to our approach that are needed to cope with the new problems are provided in most cases; the difficulties that have to be overcome are pointed out otherwise.

6.1 Position of Trains on the Track

A feature of the parking problem not present in our TUSP model, yet potentially relevant in practice, is the position of a train on the track. Even if enough physical space is left on the track, it is possible that a train cannot be parked directly on the track due to poor positioning of other trains on that track. In the example in Figure 6.1, train B has to be moved to make room for the arriving train A. To perform the repositioning of the train, a driver is needed for a few minutes. Therefore, human planners prefer shunt plans that require few repositionings. To model this aspect of the shunting problem in our algorithm, we need to determine for each service or parking activity the position on the track.

For LIFO-tracks, the ideal position of a shunt train is always at the rear-end of the track, since this position maximizes the available space on the track. This is not as

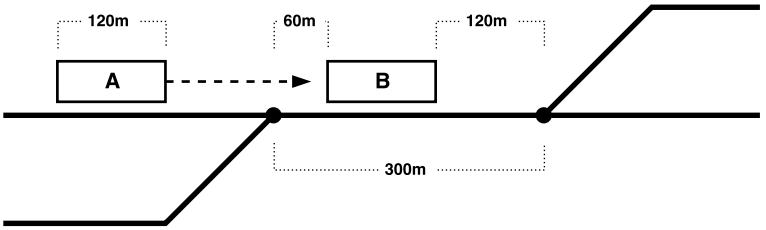


FIGURE 6.1: An example of a suboptimal parking position on the track. Although there is sufficient space on the track for both trains, train B has to be moved to allow A to arrive.

straightforward on the free tracks. Positioning a shunt train at the A-side of a free track maximizes space on the B-side, but prevents other shunt trains from parking at the A-side, unless we move the trains on the track to a different position. Furthermore, even if there already is a train parked at a free track, then it does not need to be optimal to park subsequent arriving trains as close as possible to that train, as there might be a better position that anticipates future arriving trains. An example of this can be seen in Figure 6.2(a), where train t_3 is not parked adjacent to the already parked train t_1, as that would not leave enough space for train t_4 to be parked on the track.

Computation of the position of a train depends on both the other trains parked on the track and the saw moves that use the track for the reversal manoeuvre. Conversely, the train positions affect the optimal route of a train movement. This means that the train routes are no longer independent of each other, resulting in a motion planning problem for the movements. However, including the motion planning in the local search is unlikely to yield good results, as movements themselves change frequently during the local search. Therefore, it is probably more efficient to simply compute the paths of the movements as if they are independent, allowing the computation time to be spent on parts of the problem that have a bigger impact on the overall shunt plan.

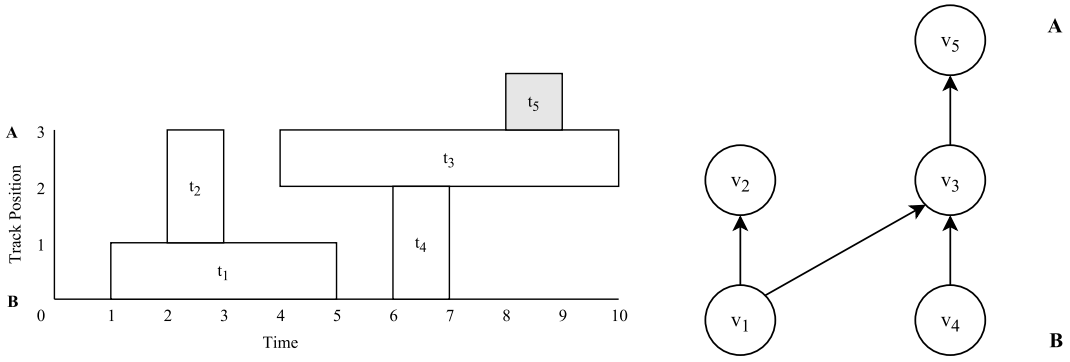
When computing the positions of trains on a track τ, we view saw moves as short intervals during which the reversing trains are parked on that track. If all parking intervals and the arrival sides of the trains are known, then we can determine the train positions on the track in linear time using a graph $G = (V, E)$ representing the adjacency relations between the parked trains. Each vertex $v \in V$ corresponds to a parking interval of a train, and we put an edge $e \in E$ between two vertices if the corresponding trains are parked adjacently on the track. Since each arrival at and departure from track τ contributes to at most one edge, the size of the edge set E is linear in the number of parking intervals.

We orient the graph G such that each arc points towards the vertex on the A-side of the track. An example can be seen in Figure 6.2(b). The resulting graph $G_{B \to A} =$

$(V, A_{B\to A})$ is acyclic, as a parked train cannot be both on the A- and on the B-side of a set of trains. $G_{B\to A}$ is then used to compute for each parked train the minimum distance to the A-side of the track that is needed to park all trains without overlap. The minimum distance $\text{dist}_A(t_v)$ to the A-side of train t_v can be computed recursively using the equation

$$\text{dist}_A(t_v) = \begin{cases} 0 & \text{if } \text{succ}_{B\to A}(v) = \emptyset \\ \max\limits_{w \in \text{succ}_{B\to A}(v)} \{\text{dist}_A(t_w) + l_w\} & \text{otherwise,} \end{cases} \tag{6.1}$$

where $\text{succ}_{B\to A}(v)$ is the set of successors of v in $G_{B\to A}$ and l_w is the length of the train t_w. In essence, we look for the path in the graph starting from v that maximizes the total length of trains corresponding to the vertices on that path. Repositioning the trains on track τ is only necessary if there exists a path in the graph of which the total train length exceeds the track length l_τ.



(a) An example train position assignment of five trains, where t_5 does not fit on the track.

(b) The corresponding graph $G_{B\to A}$.

FIGURE 6.2: An example of a track position assignment (a) and the corresponding graph used to compute the positions (b). Figure (a) represents the trains as rectangles, with on the horizontal axis the duration of the parking interval and vertically the length of the train. The longest path of adjacently parked trains, (t_4, t_3, t_5) exceeds the track length of 3. The other paths in the graph, (t_1, t_2) and (t_1, t_3, t_5) do fit on the track.

As it is not always possible to find a feasible shunt plan without repositioning some trains on a track, see Figure 6.2(a), it is desirable to minimize the number of trains that need to be shunted to a different position on the same track. However, if there already is a conflict on the track, such as a crossing or an exceedance of the track length by the total length of trains parked simultaneously, then we might as well skip the position computation, as a penalty for a conflict should dominate the maximum penalty for train repositioning on that track. Therefore, we can assume for the repositioning computation that no crossings occur on the track and that the length of the track is only exceeded by poor positioning of the trains.

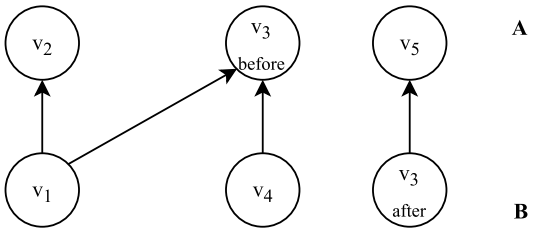


FIGURE 6.3: The graph of adjacent trains after repositioning train t_3 in Figure 6.2 just before the arrival of t_5.

When we reposition a train t_v on track τ, then the corresponding vertex v in the graph $G_{B\to A}$ will be replaced by two new vertices, v_{before} and v_{after}. All vertices w connected to v by an arc will be connected to v_{before} if t_w arrived before the reposition. Similarly, v_{after} will be connected to a vertex u if v was connected as well and t_u departed from the track after repositioning t_v. If train t_v is adjacent to some other train t_w at the time of the reposition move of t_v, then both v_{before} and v_{after} will be connected to vertex w. Therefore, repositioning trains can create a feasible train position assignment by splitting chains of adjacent trains that exceed the track length l_τ into multiple shorter chains, as is shown in Figure 6.3.

Only the arrival of a train at the track increases the length of a path in the graph $G_{B\to A}$; hence it is sufficient to limit repositioning to the moments just before train arrivals. Let $a_1, a_2 \cdots, a_n$ be the ordered sequence of the arrivals of the n trains at the track. Then we have moments $m_1, m_2, \cdots, m_n$ at which we might reposition the trains, where moment m_i takes place just before arrival a_i. For each pair of moments (m_i, m_j) with $i < j$, we will determine whether there exists a feasible track position assignment if we only consider the trains parked on the track during the interval (m_i, m_j).

We can use this information to compute a sequence of moments $M = (m_{i_1}, \ldots, m_{i_k})$ such that for each consecutive pair of moments $(m_{i_j}, m_{i_{j+1}})$ in M there exists a feasible track position assignment of the trains parked in interval $(m_{i_j}, m_{i_{j+1}})$. By repositioning all parked trains part of a path exceeding the track length at each of the moments m_{i_1} to m_{i_k} of the sequence, we get a feasible track position assignment for all trains parked on the track.

Let x_i be the number of trains parked at moment m_i on the track that are part of a path exceeding the track length. We will search for a sequence $M = (m_{i_1}, \ldots, m_{i_k})$ that both corresponds to a feasible track position assignment after repositioning on the moments m_{i_j} and minimizes the number of trains that have to be repositioned, $\sum_{m_{i_j} \in M} x_{i_j}$. A dynamic programming approach can be used to find such sequence. Let $R_{i,j}$ be the minimum number of trains that have to be shunted to a different position on the track to have a feasible track position assignment of the trains parked in the interval (m_i, m_j),

and let m_{n+1} be a dummy moment that occurs after all trains on the track have departed. Then we can use the recursive relation

$$R_{i,j} = \begin{cases} 0 & \text{if the position assignment in interval } (m_i, m_j) \text{ is feasible,} \\ \min\limits_{i<k<j} R_{i,k} + x_k + R_{k,j} & \text{otherwise.} \end{cases} \tag{6.2}$$

to construct the minimum weight set of train repositioning moments by backtracking from $R_{1,n+1}$.

The number of intervals (m_i, m_j) for which the track position assignment has to be computed is quadratic in the number of parked trains, and each computation takes linear time; hence the total running time to compute the number of track repositionings is $O(n^3)$.

6.2 Additional Resource Constraints

The service task component of the TUSP is a somewhat basic model of the crew and facility scheduling problem at the service sites. A simple, yet practical extension of the model is the inclusion of time-windows for the resources. For example, the cleaning platform might operate only between 22:00 and 7:00, thus requiring all cleaning tasks to be completed in that interval. To include such a constraint, we simply force the starting time of each service task affected to be at least the start of the time-window. Ensuring that all these service tasks are finished before the end of the interval is more difficult. Instead, a penalty can be added to the objective function for the amount of overtime. Depending on the requirements, this can either be part of the feasibility of the shunt plan — no overtime is allowed — or an optimization objective to reduce the overtime payment. Time-windows can be included in the construction of the train matching, since a train unit that has a task that must be completed in a time-window cannot depart before the start of the window.

More complex is the introduction of additional resources, such as crews for the facilities of the track-specific tasks. Crews can, for example, be assigned to multiple facilities and vice versa. This extension is rather straightforward, since it simply results in more arcs in the activity graph. The service task local search operators have to take the new constraints into account to prevent cycles and similar operators have to be used to swap the order and switch the assignment for the extra resources. However, due to the increased complexity of the problem and the additional neighborhoods, it is likely that this extension will have a significant negative impact on the running time of the algorithm.

6.3 Environmental Regulations

Due to the proximity of the service sites to urban areas, NedTrain has to comply with increasingly rigid environmental regulations. Especially sound production is an important issue, as most of the activity on the service site takes place during the night. Shunt plans can help reduce the noise disturbances by parking trains at certain tracks to block the sound or by minimizing the number of train movements in some interval.

Ideally, we would have a fast, yet reasonable accurate way to approximate the noise production at a service site. Then we could penalize a shunt plan if the noise exceeds the regulatory threshold. Unfortunately, no such function is available at NedTrain at the time of writing. In the absence of a good approximation, a poor man's solution would be to penalize the nightly train movements and reward trains parked to block the sound.

6.4 Simultaneous movements

Similar to the TUSP formulation by Lentink [22], we might allow simultaneous train movements on the service site. In this case, general railway regulations specify that the paths of simultaneously moving trains must not cross. Each train reserves the entire path for the duration of the movement.

Relaxing the restriction on the number of concurrent movements adds a very complex dimension to the train unit shunting problem, as the route of a train is highly dependent on the paths taken by other simultaneously moving trains. Furthermore, we no longer have a linear ordering on all movements. For each track we still need to define in which order the movements to and from that track occur. Otherwise, train movements will be performed as soon as possible, causing conflicts in the parking problem. Allowing simultaneous movements therefore results in a partial ordering on the movements instead of a linear one.

Other aspects of the routing extension have to be taken in consideration as well. We will list a number of questions that determine the general outline of a heuristic for the extended routing problem.

- Do we decompose the routing problem into the individual movements or will we compute the routes of simultaneous movements all at once?
- If the movements are planned sequentially, in what order do we evaluate them?

- Do we start a movement at the earliest possible starting time or can we wait until a previous movement is finished to obtain a shorter or less conflicted route?
- Do we represent the drivers explicitly in the model as resources?

Lentink computes the routes sequentially at their earliest starting time without considering the drivers. A local search approach is used to find a good evaluation order of the movements. Although the author showed that decent routes could be found in just a few seconds, improving the order of the movements with local search is not feasible for our algorithm. Our algorithm performs millions of iterations, and most search operators change one or more movements significantly.

A fast evaluation of the routing problem is thus crucial to our algorithm. This suggests that computing the routes sequentially is preferred over an integrated approach. Since there is a limited number of drivers available on site, modeling the drivers as resources will result in a more accurate approximation of reality. However, this also requires us to solve an assignment problem of movements to drivers. A simpler approach is to check during the computation of each route whether it causes the number of drivers to be exceeded, and to delay some of the involved moves if necessary. Even if we assume unlimited drivers, it is unlikely that many movements will be performed at the same time, and hence few movements are expected to be delayed in this approach.

In most cases it is preferable to start a movement as soon as possible. Even if a better route becomes available by delaying a movement, the reduction in movement duration is likely to be offset by the delayed starting time. Only if a movement has no feasible path due to the route of another movement, a delay is warranted.

The order in which the movements are evaluated can have a large impact on the overall solution quality. Unfortunately, the local search heuristic proposed by Lentink requires too much computation time. Simpler would be to flatten the partial ordering of the movements and evaluate them in that order. Movements without precedence relations that overlap in time would have to be ordered using some priority rule, which could include components such as service durations and the time table. If this order results in departure delays, an additional step would be to reorder the evaluation sequence of the movements, prioritizing movements involved in the delays, and to compute the routes once more. Using this two-step heuristic, all routes are evaluated at most twice.

6.5 Routing Extensions

Most trains maintained by NedTrain have electrical engines, collecting current from the high-voltage electrical overhead lines. However, not all service sites are fully outfitted with overhead lines. The electrical trains should not be parked on or move over tracks without overhead line. Both the parking and the routing constraints are easily implemented in the current model. In the parking search operators we only assign trains to allowed tracks. For the routing problem we can use two different routing graphs; one that includes the unpowered tracks for non-electrical trains and one that contains only the tracks with overhead lines to route the electrical trains.

Another extension that provides a more accurate representation of a service site is to model the switch positions explicitly. This is not hard to incorporate in the algorithm, as we can simply keep track of the state of a switch, updating the switch direction after each movement. To compute the duration of a route, we can, for example, use zero seconds if the switch is in the correct position and one minute if it has to be changed, instead of the fixed approximation of thirty seconds. This allows the shunt plan to use the switches more efficiently, reducing the time spent on movements. However, it does make the duration of a movement dependent on the routes of earlier movements. Since the gain of better switch usage is probably rather small, not much computation time should be spent on the optimization of the switches. The two-step approach proposed in the previous section to evaluate the movements sequentially could be applied in this case as well.

In our current model we only allow trains to perform a saw move on a track if the length of the train does not exceed the track length. However, trains can use the short pieces of track that join tracks and switches to reverse as well. For example, the railroad parts that connect the switches between track 906a and 59 in Figure 5.1 are long enough to perform a saw move, allowing a train parked on track 59 to approach track 56 from the left-hand side. To model this properly, we could assign to each approachable side of every track a tree that represents all the sequences of track parts usable when performing a saw move on that side. The tree is rooted at the track where the saw move should take place. The children of each node in the tree would be the railroad parts that are connected to the track corresponding to the parent node and on which saw moves are allowed. Checking whether a saw move is possible can be done by searching for a sequence of tracks, starting at the root of the tree, that has sufficient space for the train. This can be done efficiently with a depth-first search, as we only have to look at the children of a node if the sum of track lengths on the path to the node is less than the reversing train length and all tracks on the path are unoccupied.

Chapter 7

Conclusion

In this thesis, we have studied the problem of parking and servicing train units at service sites operated by NedTrain. The research is conducted with the purpose of determining the capacity of these sites, which is defined as the maximum number of trains that can be scheduled at the site by planners. The shunt plans are currently constructed by hand, but to answer the capacity question the planning process should be automated.

To address this issue, we presented a simulated annealing algorithm that evaluates all components of the train unit shunting problem — matching, service scheduling, parking and routing — simultaneously to construct shunt plans. This is realized by modeling the activities that take place on the service sites as nodes in a precedence graph. Our integrated search approach makes small local changes to this train activity graph to iteratively improve a shunt plan.

The performance of our proposed heuristic is compared to a tool built by the NS, named the OPG. We tested both algorithms on a number of artificial and real scenarios at one of NedTrains service sites. We have shown that our local search approach outperforms the OPG in artificial test cases, despite the years of research behind the latter [21, 23]. The proposed simulated annealing algorithm is capable of planning more trains than the OPG, even if we assume that no service tasks have to be performed. As a result, NedTrain has incorporated our local search approach in the software used to compute the service site capacity. Furthermore, a feasible shunt plan was generated by our local search approach for a large real-world scenario, indicating that the heuristic can be used to support the human planners as well. Trials are already being conducted by NedTrains long-term planners to evaluate if our algorithm can speed up their planning process. Overall, the results demonstrate that integrating all aspects of the TUSP into one local search is an effective strategy for handling the complex interactions between several NP-hard subproblems.

7.1 Future Research

Although the capacity measurement was the main goal of our research, the shunt plans generated by our algorithm can be useful to the planners as well. The long-term planners at NedTrain have expressed their interest in our local search approach to support their planning process. Practical modifications to the model needed for this use case were presented in the previous chapter. However, some of these, specifically the position on the track and simultaneous movements, require further investigation before they can be included in our approach.

Aside from the necessity to model the actual shunting process accurately, a shunt plan is only of use in real scenarios if it is also robust. Disturbances in both train arrival time and service tasks duration will often occur at the service site, and the service site operators have to adapt to these events. This means that they occasionally have to deviate from the shunt plan constructed by the long-term planners. Ideally, a shunt plan should be able to absorb most disturbances, and require only small adjustments otherwise. However, if a shunt plan is not robust, even small delays might require radical changes to the original shunt plan. Therefore, it would be interesting to investigate algorithms that include the robustness of a shunt plan in their objective. The first step would be to determine how we can express shunt plan robustness properly. There is not necessarily a single expression, as a shunt plan might be robust to some type of disturbances, and yet fail in other cases. One approach to increase the robustness is to test each shunt plan by running a number of stochastic simulations that vary arrival times and task durations, and penalize (parts of) plans that perform poorly in most simulations.

When a shunt plan becomes infeasible during the course of the day due to disturbances, the service site operator has to adapt the plan to the new situation. Preferably, the new solution would closely resemble the original shunt plan to avoid rescheduling many of the tasks of drivers and service crews. A local search algorithm such as ours could be useful to cope with this kind of problem, as it can start from the original shunt plan and iteratively improve it to regain plan feasibility. Penalties can be assigned to solutions that deviate too far from the original plan. Further research could be conducted to find a proper plan-similarity measure or other online strategies such as scheduling policies for the service operators.

The results in this study provide not only information on the performance of our algorithm, but also identify the key issue in the OPG and the underlying mathematical model by Kroon et al. [21] that limits its practical applicability. The inability of shunting a parked train to a different track in the midst of its parking interval was shown to be too restrictive to find feasible shunt plans in many cases, especially trains consist of

multiple units. To make the OPG more effective in real-world scenarios, the possibility to split the parking interval would have to be incorporated in the mathematical model. A challenging, yet interesting topic of further research would be to formulate this extension to the model without introducing an insurmountable number of additional variables and constraints.

Bibliography

[1] J. M. van den Akker, H. Baarsma, J. Hurink, M. Modelski, J. J. Paulus, I. Reijnen, D. Roozemond, and J. Schreuder. Shunting passenger trains: getting ready for departure. Technical report, Universiteit Utrecht, 2008.

[2] U. Blasum, M. R. Bussieck, W. Hochstättler, C. Moll, H.-H. Scheel, and T. Winter. Scheduling trams in the morning. *Mathematical Methods of Operations Research*, 49(1):137–148, 1999.

[3] N. Boysen, M. Fliedner, F. Jaehn, and E. Pesch. Shunting yard operations: Theoretical aspects and applications. *European Journal of Operational Research*, 220 (1):1–14, 2012.

[4] R. Bürgy, H. Gröflin, and D. N. Pham. The flexible blocking job shop with transfer and set-up times. *Journal of combinatorial optimization*, 22(2):121–144, 2011.

[5] V. Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45 (1):41–51, 1985.

[6] S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and A. Navarra. Recoverable robustness for train shunting problems. *Algorithmic Operations Research*, 4(2): 102–116, 2009.

[7] M. Dell'Amico and M. Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41(3):231–252, 1993.

[8] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[9] M. T. van Dommelen. Scheduling train service activities at service sites to determine the capacity. Master's thesis, School of Econometrics and Operation Research, Vrije Universiteit Amsterdam, 2015.

[10] C. Eggermont, C. A. J. Hurkens, M. Modelski, and G. J. Woeginger. The hardness of train rearrangements. *Operations Research Letters*, 37(2):80–82, 2009.

[11] G. W. Flake and E. B. Baum. Rush hour is pspace-complete, or why you should generously tip parking lot attendants. *Theoretical Computer Science*, 270(1):895–911, 2002.

[12] R. Freling, R. M. Lentink, L. G. Kroon, and D. Huisman. Shunting of passenger train units in a railway station. *Transportation Science*, 186(2):261–272, 2005.

[13] M. Gatto, J. Maue, M. Mihalák, and P. Widmayer. Shunting for dummies: An introductory algorithmic survey. In *Robust and online large-scale optimization*, pages 310–337. Springer, 2009.

[14] F. Glover. Tabu search-part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[15] F. Glover. Tabu searchpart ii. *ORSA Journal on computing*, 2(1):4–32, 1990.

[16] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[17] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

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

[19] D. Jekkers. Train shunt planning using genetic algorithms. Master's thesis, School of Economics, Erasmus University Rotterdam, 2009.

[20] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[21] L. G. Kroon, R. M. Lentink, and A. Schrijver. Shunting of passenger train units: an integrated approach. *Transportation Science*, 42(4):436–449, 2006.

[22] R. M. Lentink. *Algorithmic Decision Support for Shunt Planning*. PhD thesis, School of Economics, Erasmus University Rotterdam, 2006.

[23] R. M. Lentink, P.-J. Fioole, L. G. Kroon, and C. van't Woudt. Applying operations research techniques to planning of train shunting. *Planning in Intelligent Systems: Aspects, Motivations, and Methods*, pages 415–436, 2008.

[24] C.-F. Liaw. A tabu search algorithm for the open shop scheduling problem. *Computers & Operations Research*, 26(2):109–126, 1999.

[25] T. Winter and U. T. Zimmermann. Real-time dispatch of trams in storage yards. *Annals of Operations Research*, 96(1-4):287–315, 2000.