A Local Search Algorithm for Train Unit Shunting with Service Scheduling

Roel van den Broek [*1,2], Han Hoogeveen[†1], Marjan van den Akker[‡1], and Bob Huisman[§2]

[1]Utrecht University
[2]Dutch Railways

January 20, 2021

Abstract

In this paper we consider the Train Unit Shunting Problem extended with Service Task Scheduling. This problem originates from Dutch Railways (NS), which is the main railway operator in the Netherlands. Its urgency stems from the upcoming expansion of the rolling stock fleet needed to handle the ever increasing number of passengers. The problem consists of matching train units arriving on a shunting yard to departing trains, scheduling service tasks such as cleaning and maintenance on the available resources, and parking the trains on the available tracks such that the shunting yard can operate conflict-free. These different aspects lead to a computationally extremely difficult problem, which combines several well-known NP-hard problems. In this paper we present the first solution method covering all aspects of the shunting and scheduling problem. We describe a partial order schedule representation that captures the full problem, and we present a local search algorithm that utilizes the partial ordering. The proposed solution method is compared to an existing Mixed Integer Linear Program in a computational study on realistic instances provided by NS. We show that our local search algorithm is the first method to solve real-world problem instances of the complete shunting and

[*]r.w.vandenbroek@uu.nl
[†]j.a.hoogeveen@uu.nl
[‡]j.m.vandenakker@uu.nl
[§]bob.huisman@ns.nl

scheduling problem. It even outperforms current algorithms when the train unit shunting problem is considered in isolation, i.e. without service tasks. Although our method was developed for the case of the Dutch Railways, it is applicable to any shunting yard or service location, irrespective of its layout, that uses self-propelling train units and that does not have to handle passing trains.

1 Introduction

The *Nederlandse Spoorwegen* (*NS*), or *Dutch Railways*, is the largest passenger railway operator in the Netherlands. The number of passengers on the Dutch railway network will typically reach its peak in the morning and evening. During these rush hours, most of the rolling stock of NS is on the way to accommodate the flux of commuters. Outside the peak hours fewer trains are needed to serve all the passengers. The resulting surplus of rolling stock is parked off the main railway network at shunting yards such as the example shown in Figure 1.

Due to the proximity of shunting yards in the Netherlands to major stations, located in urban areas, their size, and thus their parking capacity, is very limited, reaching occupancies of up to 90%. Dense shunting yard layouts are used to exploit the available space efficiently, resulting in highly constrained train movements on the infrastructure.

The process of operating a shunting yard is called *shunting* and has the following three components: matching, parking, and routing. In the matching part we have to assign arriving train units to departing trains such that the required *train composition* — an ordered sequence of train unit types — is satisfied; here it does not matter which train unit we use, as long as it is of the correct type. The problem of finding a feasible shunting plan for the combined parking, routing and matching problem, in which every train departs on time from the shunting yard, is commonly known as the *Train Unit Shunting Problem (TUSP)*.

While solving the shunting problem is in itself already a considerable challenge (Lentink et al. (2006)), it is even more difficult for shunting yards that provide additional services. To achieve high passenger satisfaction, regular maintenance and cleaning of trains is crucial. However, due to the dense timetable and the high utilization of both railway lines and rolling stock, NS cannot afford to take trains out of service frequently. Therefore smaller service activities, such as cleaning the interior, washing, and maintenance inspections are carried out during off-peak breaks of the trains. This is done at specialized shunting yards, called *service sites*. The service activities are

constrained by the availability of resources such as maintenance crews or cleaning installations, and have to be completed before the train departs from the service site. Moreover, they have to take place at specific locations at the service site, which requires additional shunting moves. Shunting plans for service sites have to include a feasible service activity schedule, detailing for each service task when, and by which resource it will be processed. We will refer to the resulting planning problem as the *Train Unit Shunting Problem with Service Scheduling (TUSPwSS)*.

The objective in the TUSPwSS is to find a feasible solution, which is a shunting and service schedule that can be executed as planned without violating any physical or safety constraints on the railway yard. In this paper we assume that the input to the TUSPwSS is deterministic. That is, all incoming trains will arrive on schedule and there is no uncertainty in the durations of service tasks and train movements.

Constructing a conflict-free shunting and service schedule by hand is a time-consuming task, even for the experienced planners at NS. To handle the increasing number of passengers, the fleet of rolling stock of NS will be extended in the coming years. This makes this task even more complicated. Therefore, automated decision support tools need to be developed to help the human planners cope with the complex planning and scheduling problems at the service sites.

From a computational point of view, finding feasible solutions for the TUSPwSS is an extremely difficult problem. It combines several well-known NP-hard problems. The service task scheduling can be viewed as an *Open Shop Scheduling Problem* with machine flexibility (multiple identical resources), buffer and blocking constraints (shunting), and release dates and deadlines (based on the timetable). To determine whether all trains can be parked, a *Bin Packing Problem* has to be solved. Furthermore, a parked train is allowed to be reallocated to a different track if, for example, it is blocking another train's movement. Hence, the routing in the shunting plan strongly resembles sliding block puzzles such as the *Rush Hour Problem* (see Flake and Baum (2002)). Mathematical programming techniques have been thoroughly investigated for the basic shunting problem (see e.g. Lentink et al. (2006)) with varying success, but typically do not generalize well to the resource-constrained scheduling problems that we have here because of the planning of washing and cleaning; furthermore, these cannot deal with the addition of relocating parked trains. Kamenga et al. (2019) recently developed a mathematical programming formulation for the integrated version of the parking and maintenance problem, which results in a huge ILP even for small instances.

As challenging as these individual problems are, the algorithmic complexity of constructing shunting and service plans arises mainly from the strong dependencies between the components. This interaction between the different elements makes it practically impossible to effectively decompose the problem into multiple smaller, largely independent problems. Therefore decomposition approaches, which have successfully been applied to many similar complex problems, do not seem very promising here.

Our contribution

The main contribution of this paper is that we present the first algorithm capable of solving the complete Train Unit Shunting Problem with Service Scheduling (TUSPwSS) problem for real-world instances. It is based on a new partial ordering schedule of the shunting and service plan, as well as a local search algorithm that exploits the partial ordering to find solutions for the TUSPwSS efficiently. The proposed solution method is currently utilized by NS to estimate the capacity of their service sites. Moreover, pilot runs for testing its usability in daily-life operation take place at Eindhoven. Furthermore, even in comparison with a state-of-the-art mixed integer programming heuristic developed with NS specifically for the restricted case of shunting yards without service facilities (based on the work of Lentink (2006)), our local search algorithm outperforms the MIP on realistic test instances. Therefore, NS has decided to use our approach for solving the shunting problems at their yards.

In the remainder of this paper, we first provide an overview of recent literature on shunting problems in Section 2. Then, we give a problem description of the Train Unit Shunting Problem with Service Scheduling in Section 3. In Section 4 we outline our algorithm by formulating a partial ordering schedule of the shunting and service plan that captures the entire planning problem, and we propose local search neighborhoods that operate on this partial ordering. In Section 5, the solution method is tested on realistic problem instances based on the service sites operated by NS. Finally, in Section 6 we present our conclusions and indicate some directions for future research.

2 Literature Overview

The train unit shunting problem with service scheduling combines train shunting with a resource-constrained scheduling problem. As the shunting problem is the main component of the TUSPwSS, we will focus primar-

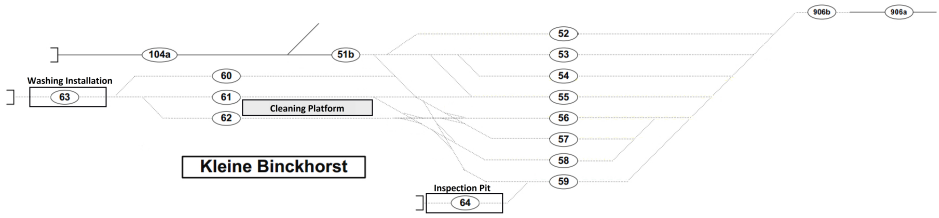


Figure 1: The "*Kleine Binckhorst*" shunting yard is operated by NS. The yard contains a washing installation at track 63, a cleaning platform between tracks 61 and 62, and an inspection pit at track 64. Tracks 52 to 59 are used to park trains. The lengths of these parking tracks range from 202 to 480 meters, and each track can contain multiple train units.

ily on literature related to passenger train shunting. Resource-constrained scheduling problems have been studied extensively over the past decades, and we will therefore only highlight literature on variants of the Job Shop and Open Shop Problems that resemble the scheduling part of TUSPwSS.

In the literature several types of shunting are distinguished. The most common one is *freight train* scheduling. Here shunting yards are used as a hub: freight trains come in from several originations at the shunting yard, where they are split and recombined such that cars with the same destination can be transported efficiently. The goal here in general is to minimize the number of cars that miss their connection and to minimize the number of movements needed to compose the trains. If we compare this to our shunting problem, then we notice several major differences. First of all, the freight train scheduling problem is mainly a sorting problem, where in contrast to our problem for each individual car the destination is given and the order of the cars in the departing trains does not matter, which implies that the matching problem does not play a role. Next, our train units can move independently, which makes it possible to move them during the night to undergo service or to get them out of the way when they block the way for other units, whereas freight cars need a locomotive to drive (or are pushed off a hump) and are usually not moved between classification and departure. Finally, shunting yards used for freight trains have a standard lay-out with several parallel classification tracks, whereas in our case the tracks on the shunting yard are more interconnected (see Figure 1). Therefore, in the remainder of this section we do not discuss the freight yard shunting problems. We refer the interested reader to the surveys by Gatto et al. (2009) and Boysen et al. (2012) and to the classification scheme by Hansmann and

Zimmermann (2008).

2.1 Passenger Train Shunting

The Train Unit Shunting Problem (TUSP) was first introduced by Freling et al. (2005) and consists of *matching* train units in arriving trains to positions in the departing trains, and *parking* these train units on a track at the shunting 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, every train unit in each arriving train is assigned to exactly one position in a departing train, such that the departing trains consist of the correct train types, and the number of times arriving trains have to be split into smaller trains is minimized. The corresponding mathematical model is solved using the standard MIP solver CPLEX. For the parking problem, the authors assume that the arriving trains are split based on the matching on the arrival track and that departing trains are combined on the departure track. Between arrival and departure, the trains are parked on a track at the shunting yard. A column generation approach, with sets of trains that can be parked on the same track as columns, is used to find a feasible parking plan. The authors propose a dynamic programming algorithm to solve the pricing problem. The routing of the trains on the shunting yard is not taken into account. They generated a shunting plan for a typical weekday at the shunting yard in Zwolle, consisting of eighty train units to be parked, in roughly half an hour.

In Lentink et al. (2006), the train unit shunting problem is extended with the subproblem of finding a route over the shunting yard for each train movement. They propose a four stage approach to construct solutions for this variant of the TUSP. First a matching of train units is determined using the algorithm proposed by Freling et al. Second they present a graph representation of the physical layout of a shunting yard to estimate the duration of moving a train from its arrival track to some parking track and back to its departure track. In the third step these estimates are included in the objective of the column generation approach proposed by Freling et al. to prefer parking assignments with low travel times. Finally, the actual routes are computed by a heuristic using the graph representation and the track occupation resulting from the previous step. The authors have shown that the time needed to generate a feasible shunting plan, including routing, for the shunting yard in Zwolle was around twenty minutes with their approach.

Instead of solving all components of the TUSP sequentially, Kroon et al.

(2006) 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 train collision constraints. Testing the model on a realistic case at the shunting 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 train collision constraints, the authors grouped these 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 for larger problems, taking several hours to complete.

Several alternative solution methods to solve the TUSP have been proposed by Haahr et al. (2015). They compared constraint programming, column generation and two-staged MIP models with a greedy construction heuristic and a reference MIP formulation on TUSP instances with LIFO tracks. Their results showed that exact techniques are outperformed by the greedy and two-staged heuristics due to excessive memory and computation time requirements.

In all these approaches, the flexibility of a shunting yard is not used to its full extent: parked trains will remain on the same track for the entire duration of their stay at the shunting yard. That is, a train is not allowed to be moved to another location once it has been parked. In contrast, we propose a heuristic that allows trains to be relocated at a different track if that is beneficial to the shunting plan, thus increasing the planning flexibility.

An integrated approach with parking reallocation has been studied by Van den Akker et al. (2008) as well. They propose a greedy heuristic and a dynamic programming algorithm to solve 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 shunting track or matching assignments at each event on the shunting 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 allow arriving or departing trains to wait at the platform to avoid conflicts at the shunting yard. Furthermore, the dynamic programming algorithm is also capable of shunting a parked train unit to a different track, resulting in much more flexibility in the shunting 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 (2006) 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 Lentink et al. (2006). The schedule for the cleaning crews is constructed last. The cleaning problem is modeled as a single machine scheduling problem without preemption, where each cleaning job needs to be finished in a time-window, and the speed of the machine varies over time to reflect the size of the cleaning crew in each shift. A mathematical model based on this formulation, in which the planning horizon is discretized into one minute blocks, is solved using CPLEX. Instead of viewing the cleaning as a single machine scheduling problem, we formulate it, together with additional service tasks, as a resource constrained scheduling problem, where the resources are only accessible from a subset of the tracks. In our approach, we do not solve it as an isolated subproblem in a sequential heuristic, as the task schedule heavily affects the parking intervals and the movements of the trains in the shunting plan.

An integral approach is used by Jacobsen and Pisinger (2011) 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 meta-heuristics, 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 shunting plans constructed by the MIP model, 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. In contrast to our work, the scope of their study was limited to task scheduling and parking. The absence of matching and routing makes it difficult to directly translate their heuristics to the TUSP with service scheduling.

In a very recent paper, Kamenga et al. (2019) present a mixed inte-

ger linear programming formulation to solve the integrated version of the parking and maintenance problem. The objective is to minimize a weighted function of the cost incurred by: cancellation of maintenance operations, delayed departures, coupling and uncoupling, routing of the train units, and the number and duration of the shunting movements. The authors work with three different sets of trains: arriving trains, intermediate trains, and departing trains; a combination of these corresponds to the route of a train unit. Incoming trains are split, if necessary, to form intermediate trains. The shunting of the trains is modeled in the trajectories of the intermediate trains; it is possible to relocate a train by connecting two or more intermediate trains. Finally, the coupling of the trains is modeled by the departing trains. Since all these possible trains have to enumerated, the size of the mixed ILP becomes huge: modeling an instance of ten trains requires approximately 2.5 million binary variables and 4.5 million constraints. The authors have tested their approach on a set of six real-life instances from the Metz-Ville station in France with up to 10 trains that must be shunted and some 25 passing trains; since there are no less than 16 tracks available, it is never necessary to park two trains on the same track. Due to the large size of the mixed ILPs, CPLEX is not able to solve any of these six instances in one hour; the remaining integrality gap varies between 7% and 24%.

2.2 Resource-constrained Scheduling

The service scheduling component of the TUSPwSS can be viewed as an Open Shop problem, where jobs, each consisting of an unordered set of operations, have to be completed to minimize an objective such as the makespan or the tardiness. Each operation has to be executed on a machine from a given set, and operations of the same job or on the same machine are not allowed to be processed simultaneously. The Open Shop problem, as well as the closely related Job Shop problem in which the operations of a job are ordered, are often represented as a *disjunctive graph*, introduced by Roy and Sussmann (1964). In the disjunctive graph model, each vertex corresponds to an operation, and precedence relations are expressed by arcs. Undirected edges are used to indicate that two operations cannot be processed at the same time. A feasible solution for the scheduling problem can be obtained by directing all the edges such that the graph becomes acyclic.

Even large instances of the Job Shop and Open Shop problem, as well as their numerous variants, can often be solved to near-optimality in reasonable time by combining the disjunctive graph representation with a local search approach. The local search relies on flipping the direction of arcs to swap the

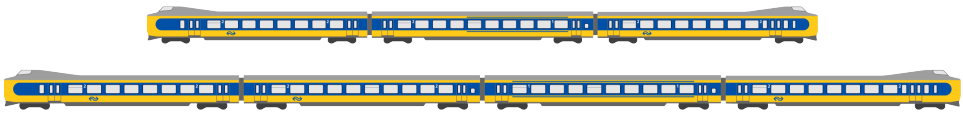


Figure 2: Examples of train units used by NS, both of the type ICM. The sub-types ICM-3 and ICM-4 indicate the number of carriages in the train.

order of operations on the same machine (Dell'Amico and Trubian (1993)), and, in the case of the Open Shop Problem, operations in the same job (Liaw (1999)), under problem specific conditions that safeguard the acyclic property of the disjunctive graph.

One interesting variant of the Job Shop and Open Shop problem that relates to the service scheduling problem at the shunting yards is to include *machine flexibility* and *blocking constraints* in the problem. An Open (or Job) Shop problem is flexible if alternative machines are available for the operations, while in a Blocking Open (or Job) Shop problem operations block their machine until the job is moved to another machine. Examples of the former property in the TUSPwSS are service sites with multiple cleaning or maintenance crews, whereas the latter occurs when a train has to wait at some service facility until another train moves out of the way. Bürgy et al. (2011) proposed local search operators that preserve the acyclic property of the disjunctive graph for the Flexible and Blocking Job Shop problems and showed that these operators can be used to find good solutions for medium-sized problem instances of this complex scheduling problem.

3 Problem Description

The problem we consider in this paper is the *Train Unit Shunting Problem with Service Scheduling (TUSPwSS)*, an extension of the *Train Unit Shunting Problem (TUSP)* as formulated by Kroon et al. (2006).

The main input of the TUSP is a timetable detailing the arrivals and departures of trains and a description of the infrastructural layout of the shunting yard. To include the service scheduling component at the service sites, the input of the TUSPwSS is supplemented with a set of resources as well as the service activities of each train unit that need to be completed before it leaves the service site.

All arrivals and departures of trains on the shunting yard are described by the timetable, and the shunting yard is assumed to be empty before

the first arrival and after the last departure. Note that a surplus of rolling stock at the start or end of the planning horizon can be modeled as early arrivals or late departures, respectively. The entries in the timetable consist of the scheduled time of the arrival or departure, the track by which the train will enter or exit the shunting yard, and a specification of the train. The rolling stock of NS consists of bi-directional and self-propelling railway vehicles that move without a dedicated locomotive. Train units are classified according to *train type* and *train sub-type*. Train units of the same type can be coupled to form longer combinations; a *train* is a coupled sequence of one or more train units. The sub-type indicates the number of carriages — and thus the length — of the train unit. For example, Figure 2 shows two train units of the same train type, ICM, but different sub-types. The ICM-3 and ICM-4 subtypes contain three and four carriages, respectively. In TUSPwSS, the level of detail of a train specification depends on whether the entry corresponds to an arrival or a departure. The timetable specifies the exact sequence of physical train units in an arrival, whereas for a departing train, it only indicates the *train composition*, which is a sequence of train sub-types. This provides the flexibility to the planners at the shunting yard to assign a train unit to any position with a matching train sub-type in the departing compositions. The scheduled arrival times in the timetable are assumed to be deterministic, i.e. we assume that all trains will arrive on time.

The service sites operated by NS consist of a set of tracks connected by switches. Tracks can either be dead-end (*LIFO-tracks*) or accessible from both sides (*free tracks*). The *length* of each track indicates the maximum total length of trains that can be parked simultaneously on that track. The duration of train movements is a function of the paths taken by the trains over the shunting yard. This function is part of the input as well. A *service site* also includes a set of resources, such as cleaning equipment or maintenance crews. Each resource can only operate on trains parked on specific tracks.

Each train unit t at the service site has a set of *service activities* that have to be completed before t leaves the site. Each service activity s for train t has a given processing time $p_{s,t}$ and requires one resource of a specific type for its entire duration. Preemption of activities is not allowed, and each resource can only process a single activity at a time. Furthermore, different service activities of the same train unit or different coupled train units cannot be performed simultaneously. We assume in this study that there are no predetermined precedence relations between the service activities.

The *objective* of the TUSPwSS is to decide whether there exists a feasible

shunting and service plan. TUSPwSS consists of the five components below.

1. **Matching**: Arriving train units must be assigned to distinct positions in departing trains such that the train unit type matches the required type in the train composition. All departing trains should leave the shunting yard on time; shunting plans with delayed departures are not feasible.

2. **Combining and Splitting**: As a result of the arrival-departure matching, arriving trains might have to be split and reassembled to form the departure composition. Splitting and combining train units takes time, up to several minutes in practice.

3. **Parking**: During its stay on the shunting yard, whenever a train is not moving, it is parked on some track on the shunting yard. The length of a track should not be exceeded by the total length of trains that are parked simultaneously on it. A train can only depart from the track it is positioned on if it is not blocked by other trains on at least one accessible side of the track. Trains are allowed to relocate during their stay at the shunting yard. Relocating a train requires an additional train movement.

4. **Routing**: For each train movement, the shunting plan should contain a path over the infrastructure. Following the notation of Gallo and Di Miele (2001), a train collision or *crossing* occurs whenever the movement of a train is obstructed by another train. Shunting plans containing crossings are not feasible. The duration of a train movement is determined by its path as well as the driving characteristics of the shunting yard.

5. **Service Scheduling**: All service activities of the train units should be scheduled such that they are completed before their corresponding train unit departs from the service site, and each resource can only process one task at the same time.

3.1 An illustration of the TUSPwSS

To illustrate the complexity of the train unit shunting problem with service scheduling, let us consider a simple scenario of three train units at the service site depicted in Figure 3. There are two arriving and two departing trains in this example, which are scheduled according to the timetable in Table 1. Note that that the sequences of train units in this table are from left to right.

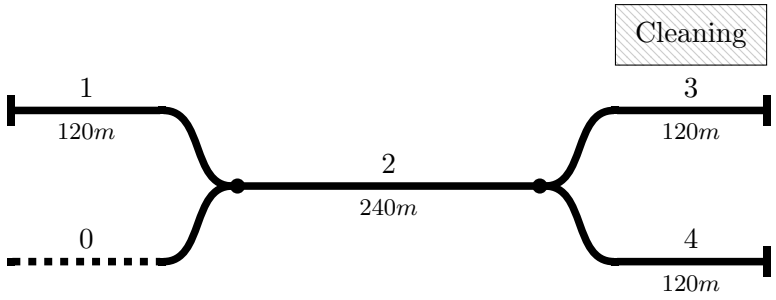


Figure 3: An example of a service site. Trains enter and exit the site over track 0 and can only be parked on tracks 1 to 4. The tracks are connected by two switches. The length of the parking tracks is displayed in meters. A cleaning platform allows internal cleaning tasks to be performed on trains positioned on track 3.

Arriving Train	Time
(1, 2)	12:00
(3)	12:45

Departing Train	Time
(ICM-3)	13:00
(ICM-4, ICM-3)	14:00

Table 1: The arrivals and departures in the example scenario. The departure trains specify the composition of sub-types instead of the train units, since the assignment is part of the matching problem. The ordering of the train units or sub-types indicates from left to right the order of the train units or sub-types in the train on the service site.

When a train moves to the left side of the shunting yard, the left-most train unit in the sequence is at the head of the train.

The train units are of the ICM type, depicted in Figure 2 and described in Table 2. Two train units are scheduled for internal cleaning, as can be seen in Table 2. In this example, we assume that every train movement takes five minutes. Furthermore, the combining and splitting of trains requires ten minutes.

Recall that to construct a shunting plan we have to decide on

- the assignment of incoming train units to positions in outgoing trains,
- how we are splitting and combining the trains,
- the order of service activities such as cleaning,
- which tracks to move the trains to,

Train Units	Type	Service Tasks
1	ICM-3 (82m)	cleaning (30 minutes)
2	ICM-3 (82m)	cleaning (30 minutes)
3	ICM-4 (107m)	none

Table 2: The train units in the example scenario.

- and the order of the train movements.

In our example, it follows from the timetable that the arriving train $(1, 2)$ has to be split into two parts, and that one of the two has to be coupled with train unit 3 to satisfy the requirements of the departing train compositions. Furthermore, with only one cleaning platform, we have to decide on which order we will clean train units 1 and 2.

Let us start the construction of a feasible shunting and service 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 departing train at 14:00. As train unit 2 is the first to depart, we schedule it to be cleaned first as well, before train unit 1. The scheduled train activities in our shunting plan are listed in chronological order in Table 3, and are illustrated in Figure 4. In this shunting and service plan, train $(1, 2)$ arrives at track 0 and moves to track 2 to be split. Then train unit 2 heads to the cleaning platform for its service task, and train unit 1 is moved to track 4 to clear track 2 for the arrival of the second train. After its arrival on track 2, train unit 3 moves to track 1 to avoid blocking the departure of train unit 2. When train unit 2 has departed, train unit 1 goes to the cleaning platform. Both train units 3 and 1 move to track 2 to be combined. Finally, the combination $(3, 1)$ departs from the service site.

This example illustrates the main complexity of the Train Unit Shunting Problem with Service Scheduling. Although the individual shunting subproblems — matching, combining and splitting, servicing, parking and routing — are seemingly easy to solve, the interaction between these components will make most shunting plans infeasible. Although parking on track 2 is possible, it blocks virtually all routes on the service site. Furthermore, poorly parked trains might require multiple reversals of direction to avoid crossings, which can easily cause departures to be delayed. The service task schedule is determined entirely by the matching, 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

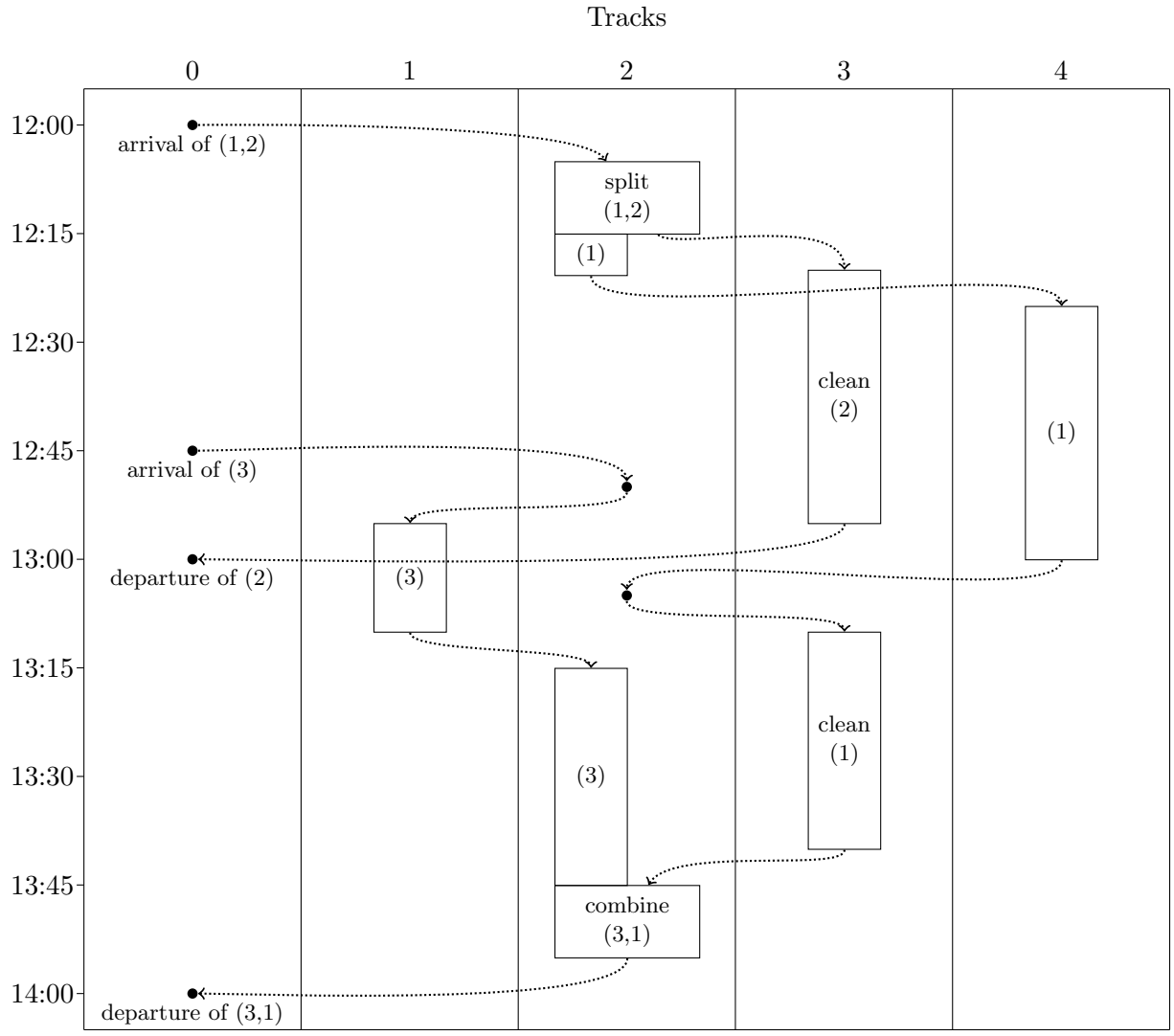


Figure 4: An overview of the positions of trains over time in the shunting plan listed in Table 3. Dotted lines represent train movements.

Start	End	Train	Activity	Tracks
12:00	12:05	(1,2)	Arrival	$0 \rightarrow 2$
12:05	12:15	(1,2)	Splitting	2
12:15	12:20	(2)	Movement	$2 \rightarrow 3$
12:20	12:50	(2)	Cleaning	3
12:20	12:25	(1)	Movement	$2 \rightarrow 4$
12:45	12:50	(3)	Arrival	$0 \rightarrow 2$
12:50	12:55	(3)	Movement	$2 \rightarrow 1$
12:55	13:00	(2)	Departure	$2 \rightarrow 0$
13:00	13:05	(1)	Movement	$4 \rightarrow 2$
13:05	13:10	(1)	Movement	$2 \rightarrow 3$
13:10	13:40	(1)	Cleaning	3
13:10	13:15	(3)	Movement	$1 \rightarrow 2$
13:40	13:45	(1)	Movement	$3 \rightarrow 2$
13:45	13:55	(3)	Combining	2
13:55	14:00	(3,1)	Departure	$2 \rightarrow 0$

Table 3: The train activities in a shunting plan for the example scenario provided in Tables 1 and 2.

infeasible solution due to the small time-window between the first arrival and departure.

4 Local Search Heuristic

To find feasible solutions for the Train Unit Shunting Problem with Service Scheduling, we propose a local search approach that includes the full problem, i.e. it integrates the matching, combining and splitting, parking, service scheduling and train movement components of the planning into a single model. Local search algorithms gradually improve some candidate solution, a shunting service plan in case of the TUSPwSS, by making small changes to it, and have been applied in the field of Operations Research with great success. Methods to create these changes are called *(local search) operators*, and the set of solutions attainable from the current candidate solution of the local search by the same operator is known as the *neighborhood*.

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 and efficient evaluation of the objective. *This is especially important as well as*

challenging for the TUSPwSS, because of the complex structure of its solutions and its tightly intertwined subproblems. We will model each activity in the shunting plan as a node in a precedence graph. We will refer to the resulting directed acyclic graph as the *activity graph*, which is a partial order schedule of the activities. The main challenge is that the graph should be updated efficiently and must remain acyclic after applying an operator.

When solving TUSPwSS, we are facing the decision problem of finding feasible shunting and service plans, where feasibility is difficult to achieve because of the high utilization factor of the yard. To alleviate this difficulty, we transform the decision problem of finding feasible shunting and service plans into an optimization problem by relaxing some of the feasibility constraints and apply local search on the resulting problem. Instead of enforcing that these relaxed constraints are respected in all solutions explored by the local search, we penalize violations of the constraints in the objective function. A shunting and service plan constructed by the local search is then feasible if and only if none of the relaxed problem constraints are violated.

We have based our algorithm on the *Simulated Annealing* framework by Kirkpatrick et al. (1983) and Černỳ (1985), which is a stochastic local search technique that has seen many successful applications to other combinatorial optimization 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 accepted 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

$$P = e^{\frac{f(a)-f(b)}{T}}, \tag{1}$$

where T is a control parameter that will be decreased during the search to accept less deterioration in solution quality later on in the process. See Section 5 for an overview of other parameters of the simulated annealing relevant to the computational experiments.

In the following we will present the objective function and the distinction between hard and soft constraints that we apply to find a feasible solution. Then we explain the representation of the solution by an activity graph and after that we discuss the local search operators. Finally, we describe the construction of an initial solution.

4.1 Optimization Objective and Constraints

Recall that we transform the decision problem of finding feasible shunting and service plans into an optimization problem by relaxing some of the constraints and penalizing violations of these constraints in the objective function. The relaxation of constraints is a trade-off between the size of the solution space and the ease of exploration of the solutions in the local search. Therefore, the decision on which constraints to relax largely defines the structure of the solution space that the local search will explore. In the remainder of this subsection, we start by providing a summary of the problem constraints. Then we motivate the relaxation choices that we make in our proposed method, and we conclude with an overview of the objective function.

We categorize the constraints of the shunting and service problem in four groups, namely

- **matching**: assign incoming train units to outgoing trains, splitting and combining the trains if necessary;
- **sequencing**: find an order for the activities that share the same service resource or movement infrastructure;
- **temporal**: ensure that trains can enter the yard directly upon arrival and depart on time;
- **parking**: park the trains without exceeding the track capacity or blocking train movements.

Constructing an assignment of arriving train units to departing trains that satisfies the **matching** constraints is not a difficult problem in itself. Moreover, any mutation of the matching — regardless of the feasibility of the resulting assignment — will likely have a large impact on the entire shunting and service plan, as it affects the parking, movement and maybe also servicing components of the solution. Therefore, we keep the **matching** constraints strict, guaranteeing that any solution will have a feasible matching.

Imposing both the **sequencing** constraints of the service tasks and the **temporal** constraints on the train departures as hard constraints makes it difficult to find a feasible schedule for the service tasks. This implies that this combination of constraints severely restricts the number of candidate solutions that can be reached efficiently during the search and hence relaxing some of the constraints will be beneficial to the search process. In our

approach we maintain the **sequencing** constraints as hard constraints, and relax the **temporal** constraints. That is, we allow the local search to construct shunting plans with delayed trains as intermediate solutions at the cost of a penalty.

Similarly, imposing the combination of **sequencing** and **parking** constraints on the train movements creates a subproblem similar to computationally difficult sliding block problems such as RushHour. Furthermore, the **parking** constraints on the track capacity alone implies that a Bin Packing problem has to be solved in each iteration, which becomes difficult in instances with a large degree of utilization of the shunting yard. Therefore, we have chosen to relax the **parking** constraints.

Violations of the relaxed **temporal** and **parking** constraints are penalized in the objective function. For a shunting and service plan, define $Delay(p)$ as the number of delayed entering and departing trains, $Crossing(p)$ as the number of crossings (i.e. collisions), and $TrackCapacity(p)$ as the number of occasions in which the combined train length of trains parked on a track τ exceed the capacity l_τ. An arrival delay will occur when an arriving train cannot move immediately from the arrival track to its parking location due to a movement of another train. These characteristics are used to quantify the weighted number of constraint violations, denoted by violations(p), of the shunting and service plan as

$$\text{violations}(p) = w_{delay} \cdot Delay(p) + w_{crossing} \cdot Crossings(p) + w_{track} \cdot TrackCapacity(p), \tag{2}$$

where each type of violation is multiplied by its corresponding weight $w > 0$, and p is feasible only if violations(p) $= 0$.

Although the expression above can be used directly as the objective of the local search, we extend the objective function with several additional terms to guide the local search to more promising regions in the solution space. The cost function minimized in the objective of our approach is

$$\text{cost}(p) = \text{violations}(p) + w_{time} \cdot \text{TotalDelayTime}(p) + w_{movement} \cdot \text{NumberOfMovements}(p), \tag{3}$$

which penalizes the severity of a delay in addition to the occurrence of violations. Furthermore, shunting plans with fewer movements are both preferred by the planners at NS and easier to improve by the local search. Therefore, we include the number of movements in the objective with weight $w_{movement}$, which is chosen small enough to never prefer a reduction in the number of movements over the resolution of a conflict.

Notation	Description
$\mathcal{A}$	Set of train activities in a shunting plan
$\mathcal{POS}$	Partial order schedule
$\mathcal{M}$	Set of movement activities, $\mathcal{M} \subseteq \mathcal{A}$
t_a	Train associated with activity $a \in \mathcal{A}$
o_a	Start location of activity $a \in \mathcal{A}$
d_a	Final location of activity $a \in \mathcal{A}$
r_a	Resource required by activity $a \in \mathcal{A}$

Table 4: Overview of the notation used to describe the shunting plans.

4.2 Solution Representation and Evaluation

Our representation of a shunting plan in the local search procedure consists of a set $\mathcal{A}$ of *train activities* and a set $\mathcal{POS}$ of *precedence relations* that defines a *partial order schedule* on the train activities. See Table 4 for an overview of the notation used in this section.

The activity set consists of four types of activities: *arrival*, *departure*, *service*, and *movement*. The precedence relations arise from the sequencing constraints. These constraints enforce that activities of the same train or on the same service resource do not overlap. Moreover, they forbid conflicts between two moving trains. Making sure that a solution satisfies these constraint boils down to sequencing activities, i.e. imposing precedence relations. Now we obtain the *activity graph*, which is a directed graph whose nodes are the activities and arcs are the precedence relations.

Each activity $a \in \mathcal{A}$ is associated to a train t_a, which is an ordered list of train units. We will refer to the set of all train movement, arrival and departure activities as the *movement activity set* $\mathcal{M} \subseteq \mathcal{A}$, with for each $a \in \mathcal{M}$ an origin o_a and a destination d_a. Note that an arrival or departure activity represents a train movement from or to the main railway network, respectively. Each service activity s is associated with a resource r_s; its location is the destination d_a of its predecessor movement. The splitting and combining of trains is modeled implicitly in the data structure by a difference in the train composition of the trains associated with subsequent activities. The representation of the example shunting plan described in Section 3.1 is shown in Figure 5.

In the activity graph, the paths taken by the train movements and the start time of the activities are not included explicitly. To keep the routing computation tractable, the local search generates activity graphs with a total ordering on the activities in the movement activity set $\mathcal{M}$, such that never

two movements overlap in time. Consequently, we can compute routing and time assignment in two steps, respectively. In the first step we strictly enforce the total ordering of the movements. We relax this restriction in the second step to allow trains to move simultaneously as long as this does not result in conflicts.

In the first step of a solution evaluation, we compute a path for each movement activity separately and determine the number of crossings and track capacity violations. To achieve this, we iterate over the movement activities according to the total order in the activity graph. For each movement activity a, we remove the train t_a from the origin track o_a, storing the number of crossings caused by this train. Then, we compute the minimum cost path of train t_a from o_a to destination d_a. Note that, due to the restriction of the movement ordering in the partial order schedule to a total ordering, all movements occur sequentially. Therefore, we can formulate the routing problem of a single train movement as a single-source shortest path problem in a graph representation of the shunting yard similar to the approach taken by Lentink (2006). The cost of a path in this graph is equal to the time it takes for train t_a to move over the path, plus the number of crossings that occur along the path times a weight λ, where λ is sufficiently large to ensure that the number of crossings is minimized. To find the shortest path we apply the $A*$ algorithm (Hart et al. (1968)), where we use the path durations in the static case without parked trains as the lower-bound heuristic on the true path cost. After computing the path of the movement, we add train t_a to the destination track d_a and update the track capacity violation count if necessary.

In the second step, we assign start times to all activities in the activity set. Due to our restriction on the partial order schedules described earlier, all train movements would be scheduled sequentially, which could result in many delayed departures in the shunting plan. To decrease unnecessary delays, we relax the precedence relations between pairs of train movements. Then, for each activity $a \in \mathcal{A}$, we compute its start time as the maximum over its release date (if it is an arrival) and the completion time of all its direct predecessors in the activity graph. More specific, when we consider movement activity a, we compute the earliest possible starting time of a such that

- a starts after the completion times of all scheduled activities, i.e. activities that have already been assigned a time-stamp, that have a train unit in common with train t_a;
- the path of a does not intersect with the path of any scheduled move-

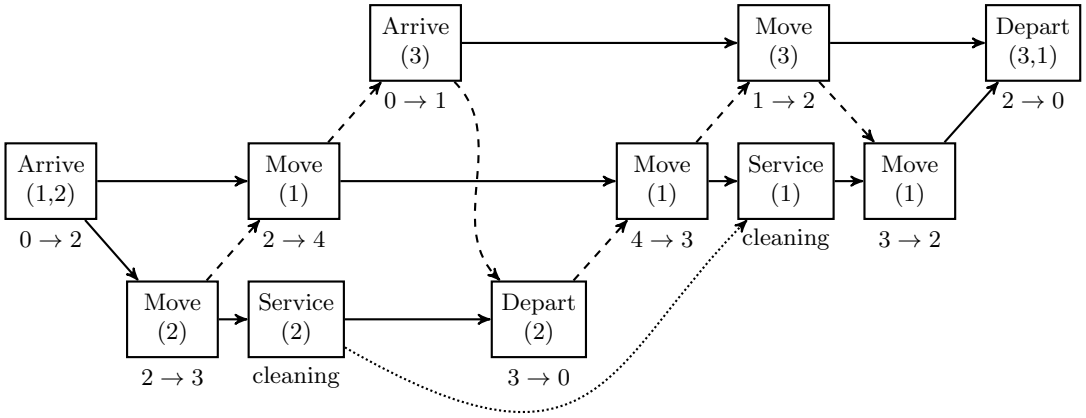


Figure 5: The partial order schedule of the shunting plan described in Section 3.1. The nodes represent train activities, with the corresponding train between parentheses, and the arcs indicate precedence relations of activities that require the same train unit (solid arcs), service resource (dotted), or movement infrastructure (dashed). The $i \to j$ notation below a node indicates a movement from track i to track j.

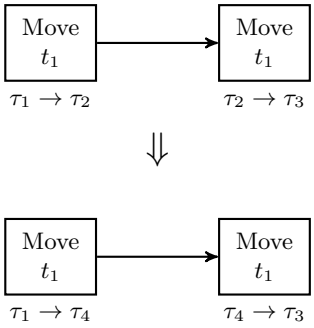
ment a' that happens at the same time;

- a does not cause additional collisions or track capacity violations.

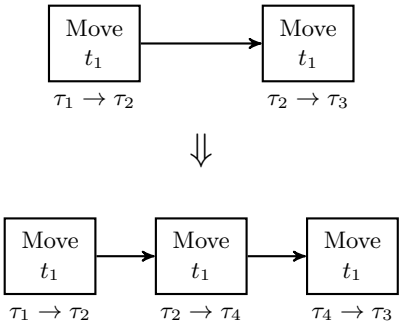
Many shortest path problems have to be solved in each candidate solution evaluated by the local search, as even small, local changes to the shunting plan can affect multiple train movements. In our approach, we only recomputed the paths of movements that might have been affected by the application of an operator in the iteration.

4.3 Search Neighborhoods

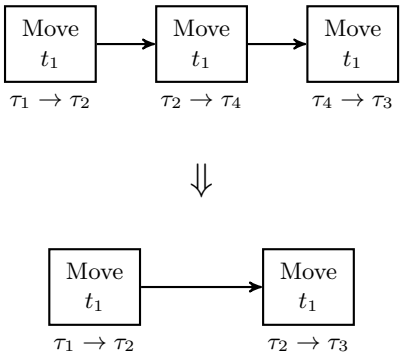
In the local search framework, new candidate solutions are selected from search neighborhoods centered around the current solution. To address the different aspects of the train unit shunting problem with service scheduling, we propose several search neighborhoods that are tailored to the different components of the planning. The corresponding operators either change the location of a train in the plan, or alter the activity graph directly by adding and removing vertices or arcs.



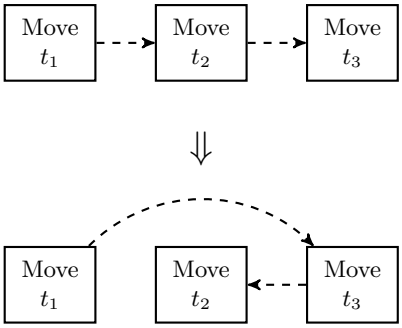
(a) Change the parking location.



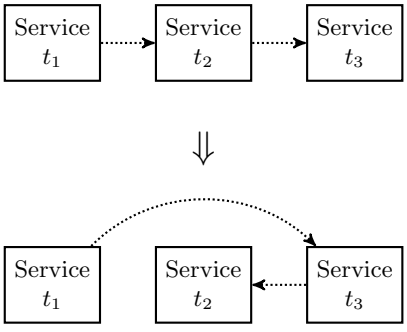
(b) Insert a train movement.



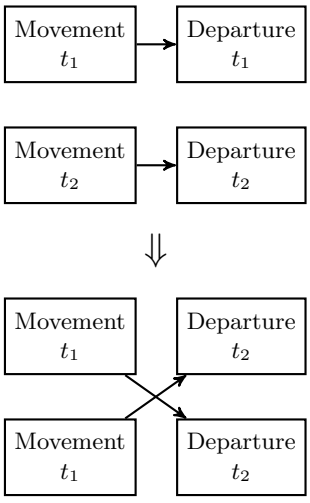
(c) Remove a train movement.



(d) Change the train movement ordering by shifting one of the movements.



(e) Change the schedule of the service activities by either assigning a service activity to a different compatible resource or changing the activity ordering within a resource.



(f) In the matching, swap the assignment of two incoming trains to outgoing train compositions.

Figure 6: Overview of neighborhoods in the proposed local search method. t_i denotes train i, $\tau_j \to \tau_k$ indicates a train movement from track τ_j to track τ_k.

To avoid deadlocks, the application of an operator to the current solution must preserve the acyclicity of the partial order schedule. As a result of the dependencies between the problem components, this means that when we change the shunting and service plan in one dimension, we also need to modify the plan in other dimensions. For example, if we change the service schedule, then the train movements have to be adapted accordingly.

We will now provide an overview of the proposed local search neighborhoods for the parking, routing, service scheduling and matching components. The splitting and combining activities follow implicitly from the other activities, and therefore have no dedicated local search neighborhoods. For each of the proposed neighborhoods that might contain solutions with cyclic precedence relations, we will show the methods implemented to restrict the neighborhood to acyclic solutions. An overview of the neighborhoods is shown in Figure 6.

Parking

Conflicts in the shunting plan such as crossings and track capacity violations can often be solved through changes in the parking location of trains. The **track assignment** neighborhood consists of all shunting plans that can be constructed by changing the track on which a train is parked. To change the location of a train, we select two consecutive movements m_1 and m_2 of the train, and assign both the destination of m_1 and the origin of m_2 to a different track, as shown in Figure 6a. If the train is split into several smaller trains after m_1, then the next train movements of all the parts have to be updated. Similarly, if m_2 is preceded by a combine activity, then all predecessor train movements of the different train parts need to be updated as well.

Train Movement

The paths taken by the trains are recomputed whenever the track occupancy changes, and as such, no local search operator is needed for the path-finding component of the routing problem. However, as we maintain a linear ordering of the movements in the partial order schedule, we can attempt to improve a shunting plan by reordering the movements. Suppose that train a is parked on a LIFO-track. If 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 **shift movement** neighborhood. The corresponding local

search operation, depicted in Figure 6d, consists of selecting a movement activity and shifting it earlier or later in the linear ordering imposed on the train movements. To ensure that the resulting shunting plan is valid, only shifts that preserve the acyclic property of the partial ordering are included in the search neighborhood.

In some cases, we want to move a train temporarily to a different track. For example, if train a has to move over track τ while train b is parked there, then one approach to resolve the planning conflict is to move train b to a different track just before the movement of a. In the partial order schedule this operation corresponds to inserting an additional movement activity for train b, visualized in Figure 6b. The **insert movement** neighborhood consists of all solutions obtainable by adding a movement activity.

Conversely, it can also be beneficial to remove redundant train movements. Suppose that a train in the shunting plan has a service activity on track τ_1, then moves to track τ_2 for parking, before continuing to track τ_3 for another service activity. If we could skip the parking and move straight from track τ_1 to τ_3 without conflicts, then we have eliminated a movement activity, resulting in more temporal flexibility for other train movements. Solutions in the **remove movement** neighborhood are constructed by removing a train movement from the solution, see Figure 6c.

Service Scheduling

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 (Dell'Amico and Trubian (1993)), the Open Shop Problem (Liaw (1999)) and their generalized counterparts (Bürgy et al. (2011)). All valid solutions that can be constructed by swapping the order of two consecutive service tasks, that are either assigned to the same resource or involve the same train, are part of the **service order swap** neighborhood, see Figure 6e. Furthermore, the **resource assignment** neighborhood contains the solutions obtained by assigning a single service activity to a valid position in the activity schedule of a different suitable resource. Observe that rescheduling service activities often requires adjusting the precedence relations of movements from and to these service activities.

Matching

The **matching swap** operator, shown in Figure 6f, changes the matching of incoming trains to outgoing departure compositions. It selects two trains t_1 and t_2 in the shunting plan of identical train composition and swaps their assignment to the departing trains.

4.4 Initial Solution

To construct a starting point for the local search, we propose a simple sequential algorithm for the TUSPwSS. 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 service tasks of the unit can be finished. Note that we can immediately abort the search for a feasible shunting plan if no perfect matching is found, as the existence of such a matching is a necessary condition for plan feasibility.

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 units are assigned in the same order to consecutive positions on a departing train,
2. their arrival time plus the sum of the durations of their service tasks is no more 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.

Based on this information, we initialize the partial order schedule:

1. The arrival and departure nodes are added to the solution;
2. For each arrival activity, we add a movement to the graph and connect it to the corresponding arrival node. Similarly, movements are added to each departure node;
3. The movements from arrivals and to departures are connected by arcs to reflect the matching, splitting and combining computed above.

In the next step, we construct a service schedule. The service activities of a train will be scheduled after it has been split, and before it will be combined in the current plan. The service tasks are scheduled by a list-scheduling strategy. Trains are sorted on increasing departure time. Service tasks of the same train are assigned to the resource that becomes available at the earliest time. If a task can be assigned to multiple resources, the resource with the currently smallest total workload is selected. Ties in the train task order are broken randomly. The service activities are then inserted into the partial order schedule and connected with arcs based on the precedence relations in the service schedule.

Next, we add movement activities to and from each service activity to the graph, as trains have to be able to reach the service facilities. The linear order of movement activities is constructed by sorting the movements by earliest starting time, based on the service task schedule.

Finally, the parking locations of the trains are assigned. For every train, we select a random track long enough to store the train for each parking time-window between consecutive movements, without taking the track occupation into account.

5 Computational Results

In this section we study the performance of the proposed local search approach on generated test cases as well as a real-world problem instance. These instances are based on two shunting yards that are considered difficult by the planners of NS due to the high degree of utilization of the yards in practice.

Preliminary tests were conducted to obtain good parameters for our local search. In all the experiments we have conducted, the local search continued searching until either a feasible solution was found, or a maximum computation time of five minutes was reached. The maximum computation time is based on preferences of NS. The control parameter T of the simulated annealing decreased exponentially in those five minutes, starting at 1 and dropping to 0.01 after 300 seconds. The weights of the objective function used in the experiments are summarized in Table 5. Delays are penalized more than the other conflicts, as these were observed to be more difficult to resolve by the local search. Furthermore, the weight of train movements is small with respect to the weights of the conflicts to ensure that conflict resolution is prioritized over the reduction of the number of train movements. Table 9 shows some results of experiments with different parameter values.

The local search procedure randomly selects a candidate solution in a neighborhood and either accepts or rejects it based on its acceptance criterion. The computations were performed on a computer with an Intel Xeon E5 3.0 GHz processor.

We will compare our simulated annealing approach with a mixed integer programming heuristic developed by NS for TUSP, i.e. only the matching, parking and routing subproblems. This tool, called the OPG, first computes the routing duration of shunting from one track to another, similar to the approach taken by Lentink et al. (2006). 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. (2006) is solved in CPLEX. In the final step of the OPG a MIP model is solved to assign starting times to all the train movements. The mathematical models in the OPG lack the flexibility of the local search algorithm to schedule service tasks or insert additional parking activities. That is, the OPG keeps a train at the same location during the entire interval between the arrival and departure of the train. As with the solution method proposed in this paper, we limit the maximum computation time of the OPG to five minutes. The OPG finished its computations within the maximum time for almost all the tested instances.

Weight	w_{delay}	$w_{crossing}$	w_{track}	w_{time}	$w_{movement}$
Value	2	1	1	0.00025	0.01

Table 5: The weights of the components of the objective function in Equation (3) used in our experiments.

5.1 Real-world Scenario

We have tested our solution method on one of the real-world instances currently planned manually at NS. The test scenario is a normal week day of twenty-four hours at the "*Kleine Binckhorst*", shown earlier in Figure 1. The Kleine Binckhorst is a medium-sized service site situated near The Hague Central Station, and consists mostly of tracks accessible from both sides. Tracks 906a and 104a connect the Kleine Binckhorst to the main railway network; parking, reversing, splitting and combining on these tracks are not allowed due to safety regulations. Tracks 52 to 63, with lengths in the range of 192 to 473 meters, are available for parking. There are two

Train type	Reversal base duration	Reversal duration per carriage
SLT	2	$\frac{1}{3}$
VIRM	4	$\frac{1}{2}$
DDZ	4	$\frac{1}{2}$

Table 6: The reversal duration of the train types in minutes. The duration consists of a base time required to transfer control and additional walking time per carriage.

Sub-type	length	Cleaning	Washing	Maintenance check
SLT-4	70	15	23	23
SLT-6	101	20	24	27
VIRM-4	109	37	24	11
VIRM-6	162	56	26	14
DDZ-6	154	56	26	18

Table 7: The train length in meters and the service task duration in minutes for each train sub-type.

dedicated service facilities: a washing machine on track 63, and a platform for internal cleaning between tracks 61 and 62. 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. The 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 reversal duration of trains and the average service task duration are listed in Tables 6 and 7. The duration of a movement is computed using the following Equation (4)

$$d_{\text{driving}} = N_{\text{tracks}} + \frac{1}{2} N_{\text{switches}}, \tag{4}$$

where N_{tracks} and N_{switches} are the number of tracks and the number of switches on the path of the movement, respectively.

The instance that we considered consists of 32 train units, arriving and departing in 23 and 21 trains, respectively. Due to the timetable, the maximum number of train units simultaneously present on the service site is 25; these train units occupy 77 percent 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. Constructing a shunt-

ing and service plan for this instance by hand usually takes more than an hour, even for an experienced planner.

We have used the simulated annealing approach described above to search for a feasible 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 trains. The shunt plan contains 88 shunting movements, of which 32 contain a reversal. The large number of reversals results in an average movement duration of 10 minutes. This means that almost 15 hours of train movements are needed in this 24-hour shunting plan. In 14 cases a parked train is shunted to a different track to make room for another train.

The shunting and service plans constructed by the algorithm differ significantly from the manually created plan. The solution of the planners clearly shows a high-level strategy of first doing the maintenance checks on tracks 56 to 59, followed by internal cleaning and washing, before parking the trains on tracks 52 to 55 until their departure. In contrast, the plan produced by the simulated annealing utilizes the tracks and resources of the service more evenly.

5.2 Generated Instances

To evaluate the performance of the proposed solution method more thoroughly, we generated problem instances for two service sites operated by NS. These instances vary in the number of train units that arrive, but resemble real-world scenarios at the service sites in all other aspects. For example, the train compositions and timetable of arrivals and departures, as well as the required service activities of train units are drawn from distributions fitted to historical data of the two service sites. The planning interval of the instances is limited to the night shift, from 6 p.m. to 8 a.m., as most activities at a service site take place during the night. Furthermore, all nightly arrivals occur before the first departure in the morning, which means that the maximum number of train units simultaneously present at the service site in a problem instance is precisely the total number of arriving train units. The train type and service task distributions used to generate the instances are shown in Table 8. The train lengths and the service durations are as in Table 7. The maximum length of composite trains in our test cases is three train units, and approximately half the arriving and departing trains are composed of two or more train units.

One of the two tested service sites is the Kleine Binckhorst. For every $k \in \{4, 6, \ldots, 32\}$, we generated 50 instances for the Kleine Binckhorst with k train units. These instances are not necessarily all feasible, particularly the

Sub-type	Arrival	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 8: The Arrival column shows the distribution of the train sub-types 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.

instances with many train units are likely to be impossible to solve. When all train units have to be cleaned internally, the planners at NS estimate the capacity of Kleine Binckhorst at roughly twenty train units during the night shift.

Since we want to compare to the OPG, which does not contain parking relocation, we investigate the impact of the parking relocation neighborhood on the performance of the local search approach. We ran the simulated annealing algorithm with and without the parking relocation neighborhood on all instances. Both variants of the local search method were run with the settings described at the start of the section.

The results of the experiments are shown in Figures 7 and 8. The proposed solution method is able to plan up to 18 train units reliably, and fails to solve instances with more than 22 train units. Removing the parking relocation operator from the local search does not result in significant deterioration of the performance. Both simulated annealing approaches show a graduate rise in computation time, requiring less than half the allotted time of five minutes to solve instances with at most 20 train units.

The similarity of the performances of the two local search variants is likely caused by the number of service activities, as these activities force the trains to move to or from service facilities, allowing the local search to solve conflicts in the parking component by changing the service schedule. To test this hypothesis, we generated similar instances without service tasks. Since only the matching, combining and splitting, parking and routing problem components remain, these instances are essentially TUSP instances. We performed the same experiments with the two simulated annealing variants as above; the results can be found in Figure 9.

In a pure TUSP instance the simulated annealing variant without the parking relocation neighborhood performs significantly worse. Without the

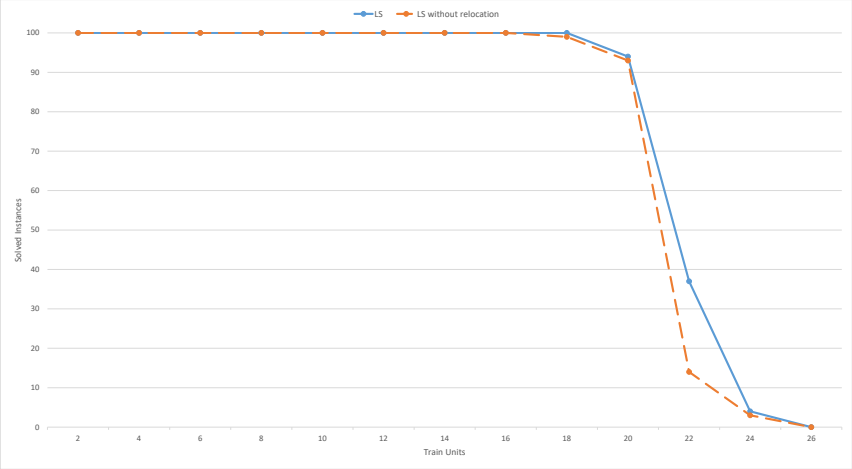


Figure 7: The number of feasible shunting plans found for each set of fifty night-shift instances of the Kleine Binckhorst. The results of both the simulated annealing with the relocation operator (*LS*) and without (*LS without relocation*) are shown.

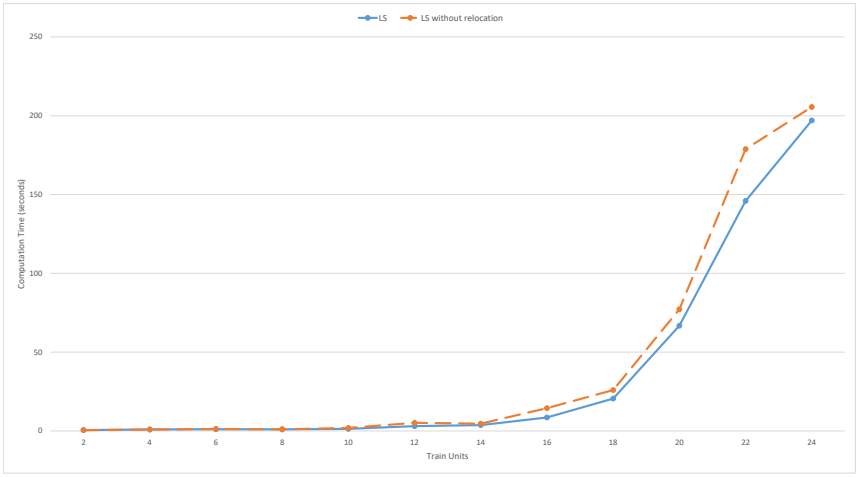


Figure 8: The average computation time in seconds of solved night-shift instances of the Kleine Binckhorst. The results of both the simulated annealing with the relocation operator (LS) and without (LS *without relocation*) are shown.

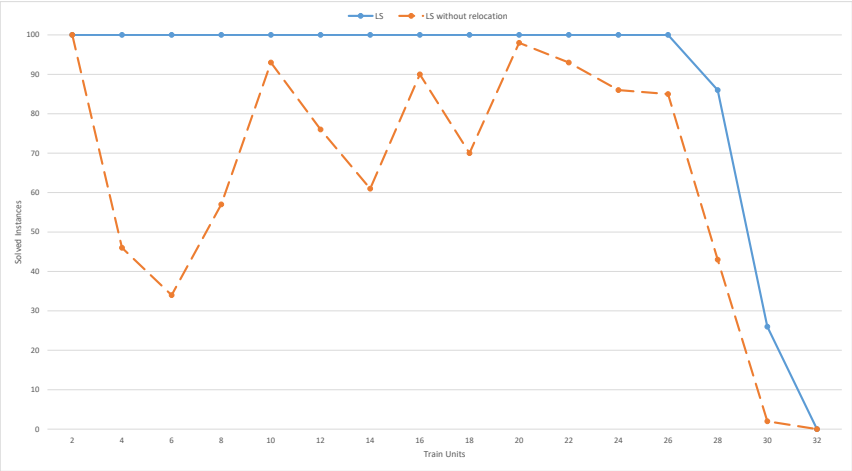


Figure 9: The number of feasible shunting plans found for each set of fifty night-shift instances of the Kleine Binckhorst without service tasks. The results of both the simulated annealing with the relocation operator (LS) and without (LS *without relocation*) are shown.

additional movements needed to reach the service facilities, trains will be parked on a single track for their entire stay at the shunting yard, making it difficult to resolve some conflicts. For example, suppose we have an instance with the arriving train (a, b), where sub-type$(a) \neq$ sub-type(b), and a departing train composition (sub-type(b), sub-type(a)). Then, without additional movements it is not possible on the Kleine Binckhorst to split and combine train (a, b) into the proper departure composition, as splitting or combining is not allowed on the arrival track. In instances with a sufficient number of train units of each type, this type of conflict is often easily resolved by changing the matching. However, smaller instances are more likely to be impossible to solve without the parking relocation neighborhood, as can be seen in Figure 9.

In general, by removing the service tasks — using the service site only as a shunting yard — the proposed solution method is capable of finding feasible shunting plans for more train units, reaching an 85% utilization of the parking capacity of the service site in some cases. This suggests that service scheduling and the train movements to and from the facilities are a major bottleneck in the earlier experiments.

Another set of instances for the Kleine Binckhorst was generated to compare the local search algorithm with the OPG, the ILP-tool developed by NS. Similar to the previous experiment, the instances do not contain service activities. However, instead of only modeling the night shift, the instances in this set span an entire day, where trains arrive in the evening and at the end of the morning, and departures occur mostly before the morning and evening rush hours. The performance of the solution methods are shown in Figure 10.

The differences between the two local search variants are similar to the results shown in Figure 9. The OPG shows results resembling those of the local search without the parking relocation neighborhood, and is outperformed by the local search with relocation. So our algorithm outperforms the OPG for TUSP. The similarity between the results of the OPG and the local search without relocation can be explained by the mathematical model in OPG, which does not have the flexibility provided by the parking relocation neighborhood.

To test the proposed solution method for other service site layouts, we conducted similar experiments with instances generated for service site *OZ*, located near Utrecht Central Station. In contrast to the Kleine Binckhorst, most parking tracks of OZ are last-in-first-out tracks, see Figure 11. Trains will arrive and depart via track 117. The connection of track 117 to the main railway network prohibits parking, reversing, splitting and combining

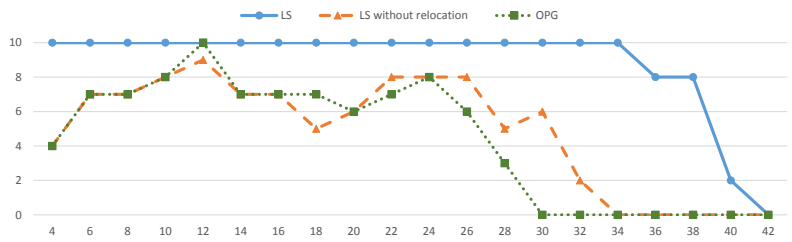


Figure 10: The number of feasible shunting plans found for each set of full-day instances without service tasks of the Kleine Binckhorst. The results of the simulated annealing variants with the relocation operator (*LS*), without relocation (*LS without relocation*) and the ILP-based OPG are shown.

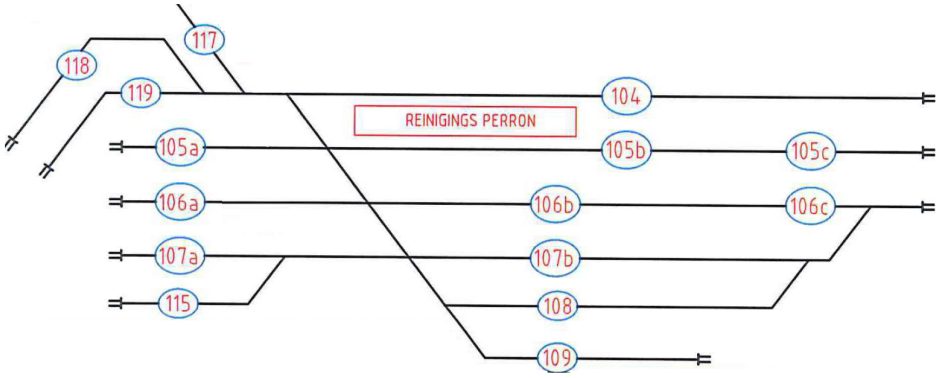


Figure 11: The service site *"OZ"* operated by NS.

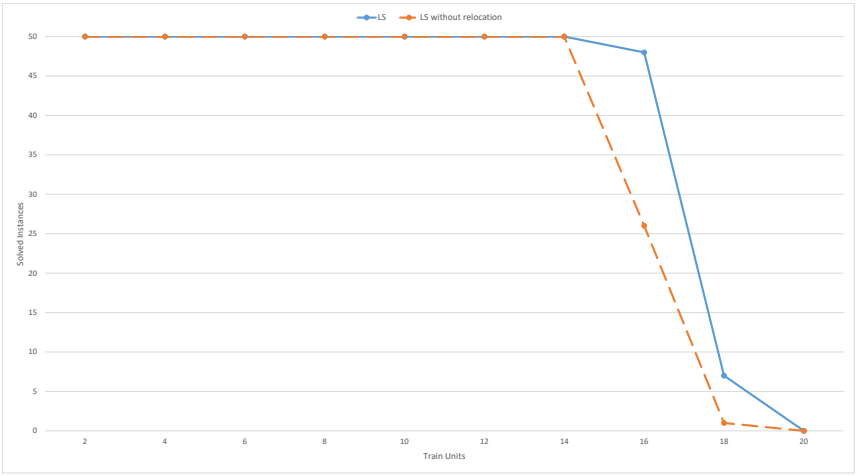


Figure 12: The number of feasible shunting plans found for each set of fifty instances of service site OZ. The results of both the simulated annealing with the relocation operator (*SA*) and without (*SA without relocation*) are shown.

of trains on it. Parking is possible on all other tracks visible in Figure 11. The cleaning platform is accessible from tracks 104 and 105b. Instances for the night shift are generated using the same parameters as for the Kleine Binckhorst, with the number of train units ranging from 4 to 22. The same service tasks are assigned to the train units, with the exception of washing activities due to the absence of a washing installation. The results, shown in Figure 12, confirm those of the experiments with the Kleine Binckhorst, as for TUSPwSS the local search with relocation performs slightly better than without the relocation operator.

In addition to the relocation neighborhood experiments we investigated the impact of the values of the local search parameters described in Table 5. We ran the local seach with seven different parameter settings on a subset of the instances used in the experiments in Figure 7. The results are listed in Table 9. Parameter setting 1 describes the parameter values used in the other experiments in this section.

The experiments show that penalizing the movements is necessary to solve the majority of the instances, as the parameter settings without move-

	weights					results	
set	w_{delay}	$w_{crossing}$	w_{track}	w_{time}	$w_{movement}$	feas.(%)	comp. (s)
1	2	1	1	0.00025	0.01	92	94
2	2	1	1	0.00025	0	36	182
3	2	1	1	0.00025	0.1	84	69
4	1	1	1	0	0	40	109
5	1	1	1	0	0.01	54	165
6	1	1	1	0.00025	0.01	48	160
7	3	1	1	0.001	0.01	66	98

Table 9: The percentage of solved instances and average computation time for different parameter values of the local search algorithm.

ment penalties (settings 2 and 4) perform significantly worse than the other settings. Furthermore, a small incentive to prefer the resolution of delays over other conflicts increases the likelihood that the local search finds a feasible solutions.

6 Conclusion and Further Research

In this paper, we have studied the problem of planning the parking and servicing train units at service sites operated by NS. The research is conducted with two purposes, firstly to support human planners with the construction of feasible shunting plans for the service sites, and secondly to improve the capacity estimates by the management of NS.

We have introduced the *Train Unit Shunting Problem with Service scheduling (TUSPwSS)*. Although the Train Unit Shunting Problem has been studied, there are no practical algorithms that include the resource-constrained scheduling of service tasks.

We have presented a local search approach to find feasible plans for TUSPwSS. *This is the first algorithm capable of constructing feasible plans for real-world instances of the full shunting and service scheduling problem.* The solution method consists of a plan representation that models the precedence relations between the scheduled activities, as well as local search neighborhoods exploiting the partial ordering. We have benchmarked our approach on both generated and real-world instances of service sites operated by NS. The experiments showed that our solution method is capable of solving shunting problems on service sites with varying infrastructural

layouts within a few minutes.

Moreover, we compared our algorithm to the OPG, a decision support tool based on state-of-the-art mathematical programming models that has been developed by NS. In the solutions of the OPG trains are parked at a fixed place during their stay at the yard. The OPG does not include service scheduling. Therefore, we included instances without service scheduling in our experiments.

Comparison with the OPG showed that our local search algorithm is capable of solving harder instances than the ILP-based OPG. Since an important difference is the possibility for relocation, we decided to investigate this aspect further by also running our algorithm without relocation. These experiments demonstrated that the flexibility to move a train to a different track during parking is essential to find feasible plans. Note that in TUSPwSS the possibility for relocation is obtained partly from the service schedule, which forces trains to move to service facilities.

The real-world scenario illustrated that the local search approach is a valuable tool in the planning process at NS by providing human planners with feasible solutions, drastically reducing the time needed to construct good plans for service sites. The local search method is currently being used by NS to obtain a good estimate of the capacity of their service sites by generating realistic problem instances for varying numbers of train units, and checking for which number the local search algorithm can still consistently find feasible solutions.

For the sake of brevity we have limited the scope of this paper to shunting yards. However, NS is currently performing a pilot for application of the local search in the daily operation at the railway node Eindhoven. This node encompasses both the major station Eindhoven and two nearby shunting yards. The goal of the pilot is to create an integrated shunting plan of the complete node. To model the railway traffic properly, we have to include through trains, which are trains that move through the station without going to a yard. As the schedule of the through trains is predetermined by the timetable, we model them as train movement activities with fixed routes, release dates and deadlines in our shunting plans. Furthermore, we have to adapt the local search to more sophisticated movement constraints, such as minimum headway between movements over a track or switch, and non-empty shunting yards at the start and end of the planning horizon.

Preliminary results from the pilot show that in most cases we are able to find feasible shunting plans for real-world problem instances on these large and complex locations. Even when the local search fails to find a feasible solution, the number of conflicts in the final solution is reduced to such an

extend that human planners can resolve remaining conflicts in a fraction of the time that it would take them to construct a shunting plan from scratch. Although the required computation time on these instances ranges from 15 minutes to an hour, we hope to reduced this by further tuning the parameters of the local search.

The pilot shows that the main strength of our local search approach lies in its flexibility, as it is easily adaptable to the often complex constraints that arise in real-world problem. This is especially appreciated by the practitioners at NS, as we are able to quickly incorporate their planning preferences as well.

Opportunities for further research present themselves both in extending the scope of the model, and in coping with the complications arising from the actual operation. With respect to the first, a major component of the planning process that we have thus far ignored in this paper is the personnel planning, i.e. assigning activities such as train movements and maintenance checks to drivers and mechanics, respectively. The physical size of shunting yards often necessitates the inclusion of walking durations between different locations, especially for the train drivers. To further support human planners, we are currently developing methods that construct feasible service schedules for the personnel as well.

As to the latter, a shunting plan can only be implemented if it is robust to the everyday disturbances in operations. Both the arrival times of trains and the durations of service tasks will often vary in practice, and the service site operators have to adapt to these events. This means that they occasionally have to deviate from the shunting and service plan constructed by the planners. Ideally, the plan should be able to absorb most disturbances and require only small adjustments otherwise.

To address this issue we have started research to improve the robustness of shunting plans. As a first step, we focus on minimizing the likelihood of delayed departures in the shunting and service plans when the arrival time of trains and the processing time of service tasks are uncertain. In Van den Broek et al. (2018) we investigate the predictive power of robustness measures, i.e. functions that can be computed efficiently from the schedule and are designed to represent properties of robust schedules. We focus on measures that represent the likelihood of delays in shunting plans. For the robustness measures that are strongly correlated with the delay likelihood, we show in Van den Broek et al. (2019) that our local search finds more robust shunting and service plans with only little computational overhead by including the robustness measures in the objective function.

When a shunting and service 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 shunting plan to avoid rescheduling many of the tasks of drivers and service crews. A variant of the local search algorithm presented in this paper could be useful to cope with this kind of problem, as it can start from the original shunting plan and iteratively improve it to regain plan feasibility. Penalties can be assigned to solutions that deviate too much from the original plan. Additional research has to be conducted to find a proper plan-similarity measure or other on-line strategies such as scheduling policies for the service operators.

References

van den Akker, J. M., H. Baarsma, J. Hurink, M. Modelski, J. J. Paulus, I. Reijnen, D. Roozemond, J. Schreuder. 2008. Shunting passenger trains: getting ready for departure. *Proceedings of European Study Group Mathematics with Industry* **63**.

Boysen, N., M. Fliedner, Jaehn F., Pesch E. 2012. Shunting yard operations: Theoretical aspects and applications. *European Journal of Operational Research* **220**.

van den Broek, R., J.A. Hoogeveen, J.M. van den Akker. 2018. How to measure the robustness of shunting plans. *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

van den Broek, R., J.A. Hoogeveen, J.M. van den Akker. 2019. Finding robust shunting plans. *10th Triennial Symposium on Transportation Analysis, (TRISTAN 2019)*.

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

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

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

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

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

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

Gatto, M., J. Maue, M. Mihalák, P. Widmayer. 2009. *Shunting for Dummies: An Introductory Algorithmic Survey*. Springer Berlin Heidelberg, Berlin, Heidelberg, 310–337. doi:10.1007/978-3-642-05465-5_13. URL `https://doi.org/10.1007/978-3-642-05465-5_13`.

Haahr, J. T., R. M. Lusby, J. C. Wagenaar. 2015. A comparison of optimization methods for solving the depot matching and parking problem. Tech. rep., Erasmus Research Institute of Management.

Hansmann, R.S., U.T. Zimmermann. 2008. *Optimal Sorting of Rolling Stock at Hump Yards*. Springer Berlin Heidelberg, Berlin, Heidelberg, 189–203. doi:10.1007/978-3-540-77203-3_14. URL `https://doi.org/10.1007/978-3-540-77203-3_14`.

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

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

Kamenga, F., P. Pellegrini, J. Rodriguez, B. Merabet, B. Houzel. 2019. Train unit shunting: Integrating rolling stock maintenance and capacity management in passenger railway stations. *RailNorrköping 2019. 8th International Conference on Railway Operations Modelling and Analysis (ICROMA), Norrköping, Sweden, June 17th–20th, 2019*. 069, Linköping University Electronic Press, 528–547.

Kirkpatrick, S., C. D. Gelatt, M. P. Vecchi. 1983. Optimization by simulated annealing. *Science* **220**(4598) 671–680.

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

Lentink, R. M. 2006. Algorithmic decision support for shunt planning. Ph.D. thesis, School of Economics, Erasmus University Rotterdam.

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

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

Roy, B., B. Sussmann. 1964. Les problemes d'ordonnancement avec contraintes disjonctives. *Note DS* **9**.