A Comparison of Optimization Methods for Solving the Depot Matching and Parking Problem

Jørgen Thorlund Haahr[1], Richard Martin Lusby[1], and Joris C. Wagenaar[2]

[1]Department of Management Engineering, Technical University of Denmark, Produktionstorvet 424, 2800 Kgs. Lyngby, Denmark, email: jhaa@dtu.dtk, rmlu@dtu.dk
[2]Rotterdam School of Management, Erasmus University , Technology and Operations Management, Burgemeester Oudlaan 50, 3062 PA, Rotterdam The Netherlands, e-mail: jwagenaar@rsm.nl

October 15, 2015

Abstract

We consider the Train Unit Shunting Problem, an important planning problem for passenger railway operators. This problem entails assigning physical train units to scheduled train services in such a way that the resulting shunting yard operations are feasible. As such, it arises at every shunting yard in the railway network and involves matching train units to arriving and departing train services as well as assigning the selected matchings to appropriate shunting yard tracks. We present a comparison benchmark of multiple solution approaches for this problem. In particular, we have developed a Constraint Programming formulation, a Column Generation approach, and a randomized greedy heuristic. We compare and benchmark these approaches against slightly adjusted existing methods based on a a Mixed Integer Linear Program, and a Two-Stage heuristic. The benchmark contains multiple real-life instances provided by the Danish State Railways (DSB) and Netherlands Railways (NS). The results highlight the strengths and weaknesses of the considered approaches.

Keywords: *Passenger Railway Optimization, Shunting, Matching, Parking*

1 Introduction

Passenger railway operation is an important mode of transportation in many countries. Commuters depend on a safe, reliable and timely operation that requires careful planning of trains, personnel and infrastructure. Planning

the operations of a railway operator includes, among other things, *i)* determining a timetable, stipulating arrival and departure times of the train services to be operated, *ii)* creating a rolling stock schedule, specifying a feasible fleet circulation and *iii)* constructing a crew plan, assigning train personnel to operate the trains. Due to the complexity of each the underlying optimization problems, the planning problems are usually resolved sequentially and in isolation.

A rolling stock schedule indicates which individual train units are allocated to each of the timetabled train services. If multiple units are assigned to the same train service, this results in a given train *composition*. Train services can, and do, have different compositions since the allocation of train units to services is done in such a way that the passenger demand is matched as closely as possible. The aim is typically to meet the passenger demand forecast, while not using more train units than is necessary. Whenever the composition changes between two successive train services, either a train unit is taken out of service or a train unit is brought into service. In the first case, the uncoupled unit must be *shunted* to the station's *depot* where it awaits its next service, while in the second case a unit must be retrieved from the depot and coupled to the train service. In both cases *shunting movements* are induced.

Shunting movements are often not considered when planning the rolling stock schedule. It is usually assumed that these can be resolved in a post-processing phase. The assumption is that the capacity as well as the infrastructure layout of any depot (i.e. *shunting yard*) is sufficient to cater for the induced shunting movements. In this paper we challenge this basic assumption. For railway networks where depot capacity is scarce, such as the suburban railway network in Copenhagen, it is not always possible to perform the shunting movements induced by a given rolling stock schedule. Furthermore, planning such movements is not a trivial problem. This is especially true at larger stations that have many shunting movements occurring over the course of a day and many depot tracks of different lengths on which to park train units not in service. Effective methods for finding feasible depot plans are essential.

In this paper, we present a comparison of optimization methods for determining whether the scheduled shunting movements at a given depot are feasible with respect to the number of depot tracks available. We term this problem the Train Unit Shunting Problem (TUSP) and benchmark several previously proposed methods from the literature on both realistic and artificial problem instances. In addition, we also test and compare three novel approaches for solving this problem. The first is a constraint programming approach, the second is column generation based approach, and the third is a greedy randomized heuristic approach. We view the TUSP as a feasibility problem only, as it ultimately determines whether or not a given rolling stock schedule is feasible. The majority of the operational cost is

incurred in the rolling schedule and this would almost never be changed in order to improve the combined objective of the TUSP problems at each of the depots. It is important to simply know whether the induced shunting movements are feasible with respect to the depots. We note that, after confirming feasibility of the shunting movements, the TUSPs can be resolved with an appropriate objective function.

The set of shunting movements at each of the depots in the railway network can be deduced from a given rolling stock schedule. Scheduling the rolling stock is, however, beyond the scope of this paper; it is assumed to be given as input. From a rolling stock schedule all arrivals and departures from depots are implicitly specified. Railway operators typically have fleets of train units of different types. Two different unit types typically differ in their respective physical characteristics, e.g. length and passenger capacity. We assume rolling stock units of the same type to be interchangeable. The depots of two different stations are also assumed to be independent of each other. All shunting movements at a given depot are confined to that depot and have no impact on the shunting movements of other depots. A feasible solution to the TUSP, termed a *shunting plan*, must satisfy the following two constraints: the total length (or capacity) of each individual depot track is not violated at any given time, and no unit ordering *conflicts* are present. A conflict occurs when the arrival of a unit at a depot track blocks the departure of a unit from the same track. Depot tracks are assumed to function as last-in first-out (LIFO) queues, meaning the last unit to arrive at a depot track must be the first to leave. In reality, open ended depot tracks are allowed; however, such cases will also have ordering restrictions that must be obeyed. In this paper, any open-ended track is operated as a LIFO queue.

The TUSP entails finding a feasible shunting plan. However, in the absence of any possible solution, proving that no solution exists is equally as important. In such situations, a new rolling stock schedule must be found. The emphasis in this paper is on determining whether a solution exists. As mentioned above, no cost-structure is used to distinguish or rank distinct feasible solutions to a given instance of the TUSP.

We compare previously proposed methods and devise new methods to tackle the problem using different optimization techniques. These include: Constraint Programming, problem decomposition, Column Generation, and Mixed Integer Linear Programming with delayed constraint generation. Some of the proposed approaches are exact solution methods, which find one feasible solution (if it exists) or prove that no solution exists. The other methods are heuristic approaches that strive to quickly find any feasible solution by searching subsets of the entire solution space and consequently cannot prove infeasibility.

This research makes several contributions to the existing literature in this field. These include the following. First, we describe several efficient

initial infeasibility checks for the TUSP. Secondly, we develop three novel methods for the TUSP: *i)* A constraint program formulation, *ii)* a column generation approach, and *iii)* a randomized greedy construction heuristic. We compare the performance of these against Mixed Integer Program (MIP) approach and a two-stage decomposition approach; two methods that have been proposed in the literature, for reference. Due to the high memory consumption of the MIP approach, we investigate the potential of delayed constraint generation for this method. A *track and unit type* decomposition approach is also presented that reduces a TUSP instance into several smaller and independent TUSP sub-problems. Furthermore, we discuss and present rolling stock platform parking, which is an interesting extension used by several railway operators in practice in order to circumvent capacity issues. Finally, all methods are benchmarked on TUSP instances from multiple railway operators residing in different countries. A few additional realistic benchmarks are performed in order to test different aspects of the presented methods.

The remainder of this paper is structured as follows. First, in Section 2 we give an overview of related literature and highlight the main differences to our contributions. In Section 3 we present the problem description. A number of polynomial solvable feasibility checks are discussed in Section 4 before introducing the solution methods in Section 5. The problem instances are presented in Section 6 followed by the benchmark of the solution methods. Finally, we conclude and give some remarks on further research in Section 7.

2 Literature overview

To our knowledge, the TUSP was first introduced by Freling et al. [1]. Other authors have considered different variants of the same problem, including additional constraints and decisions such as maintenance operations or station routing. In some cases the train matching is given as input and not part of the problem. With the exception of Kroon et al. [2], all studies do not integrate the matching and parking problem, but solve them separately. A cost structure is often used to rank different matching and parking assignments. We, however, focus on the core matching and parking problem and do not differentiate between distinct solutions.

Freling et al. [1] consider the problem of parking train units overnight at a shunting yard in such a way that each unit can be retrieved when needed during the operations of the following day. Any feasible parking must ensure that train units of different types do not block each other when departing the yard the next day. The problem is decomposed into two smaller sub-problems: a matching problem and a parking (or track allocation) problem. The first entails matching arrival and departure services at the shunting yard

under consideration. A solution to this problem hence also stipulates the train services each physical unit will perform. This problem is formulated as a MIP and solved using a commercial solver. The parking problem, on the other hand, determines how to park the assigned matchings on each of the tracks at the yard such that the track capacity is never exceeded and such that the movements associated with the matchings are conflict-free. The authors model this as a set partitioning problem with side constraints and solve it using a heuristic column generation procedure. A corresponding MIP approach was not considered nor compared. The proposed approach was tested on one instance from Netherlands Railways and results were found within 20 to 60 minutes of computation time. In contrast to our work, the authors adopt a cost-structure to both problems in order to rank the solutions. We compare our methods with a variant of this approach in Section 6.

The work of Lentink et al. [3] extends the work of Freling et al. [1], where a four-step approach is proposed to solve the matching and parking problem. The problems are still solved independently; however, the parking problem is extended to include more practical aspects, e.g. cleaning and maintenance of units, as well as the routing costs incurred from the shunting yard to the station platforms. The small problem instances are solved quickly, but the larger instances require at least 700 seconds of computation time.

A dynamic programming based heuristic approach for the TUSP is proposed by Haijema et al. [4]. The matching and parking problems are solved sequentially and in isolation. To reduce the problem size the authors propose a rolling horizon technique to solve the problem. A realistic test case from the railway station Zwolle in the Netherlands is used to analyse the performance of the algorithm. A 24 hour period is considered in which 45 units arrive and 55 units depart. The shunting yard has 19 tracks and a total capacity of 4000 metres. Solutions to the problems are found quickly and the results are promising; however, only a single instance is considered.

In contrast to Freling et al. [1], Lentink et al. [3], and Haijema et al. [4], we propose methods that integrate the matching and parking problems. In addition, we present and discuss the possibility of parking rolling stock units at platforms at the end of the planning horizon. Different models of the problem are presented and compared considering multiple different problem instances. Finally, as has been mentioned, we consider the TUSP to be a feasibility problem and hence do not differentiate between different, feasible plans.

Solving the TUSP without some form of matching/parking separation has been proposed by Kroon et al. [2]. The authors essentially extend the work of Freling et al. [1] and propose a large MIP formulation that simultaneously solves the matching and parking problems. The authors propose to minimize the number of splitted compositions, and in addition try to keep the depot tracks as homogeneous (w.r.t. unit type) as possible. In addition, they also consider conflict-cliques in order to reduce the large number of

conflict constraints. In contrast, in our work we accommodate this problem by adding conflict constraints on the fly in the Branch-and-Bound (B&B) framework. Practical restrictions that include how to handle depot tracks that can be approached from both sides are described. Two stations from the Dutch Railway network form the computational study, where instances with up to 125 train units of 12 different types are considered.

Jacobsen and Pisinger [5] present three different heuristics to solve a variant of the TUSP that includes maintenance scheduling. The model is tested on small instances and the runtimes are low. Internal rearrangements are permitted if one unit is blocking another unit. The model is not tested on problem instances from practice.

3 Problem description

In this section we give a formal problem description for the TUSP and introduce some general notation that is used throughout the paper. The core problem is to create a feasible shunting plan or prove that no such plan exists. A feasible shunting plan matches all initially parked units at the depot, as well as any units that arrive during the day, with compatible departures. Any unmatched unit must remain parked at the depot. Unmatched units reside in inventory until the end of the planning period. In a feasible matching all departures must be covered, otherwise the TUSP is infeasible. Given a feasible matching, the unit assignments to depot tracks must be conflict-free.

We term all units initially parked at the depot and all units that arrive during the day as *arrival* events. A specific unit type and known arrival time are associated with each arrival. The arrival time of initially parked units is assumed to be the start of the planning horizon t_0. Likewise, departing units are termed *departure* events. A departure has a known departure time and a required unit type. For the sake of simplicity, a departure is defined for all units remaining in the depot at the end of the planning horizon, t_∞. Consequently, a feasible matching covers all arrival events and all departure events. In other words, initially parked and arriving units are matched to either a compatible departure or assigned to stay on some track in the depot at time t_∞.

The matching and parking assignment must satisfy two types of constraints. First, the capacity on each individual depot track may not be violated at any time. In other words, the total length of all train units residing on a track at any time may not exceed the length of the track itself. It suffices to ensure that this holds whenever a train unit arrives at the depot. Second, all tracks must be processed in a LIFO order, i.e the last parked unit on a track must be the unit that leaves first. A unit cannot leave a track (for a departure) if a different unit has arrived in the meantime and is staying on the same track. We assume that an assigned track is occupied

Event	Type	Time
Arrival	a_1	12:00
Arrival	a_2	12:30
Arrival	b_1	13:00
Arrival	c	13:30
Arrival	b_2	14:00
Departure	b	15:00
Departure	c	15:30
Departure	a	16:00

Table 1: Example list of events in a problem instance. Note, that a_1 and a_2 (and b_1 and b_2) denote the same type, only different physical units.

Matching 1	Matching 2
(b_1, b)	(b_2, b)
(c,c)	(c,c)
(a_1, a)	(a_1, a)
(a_2, inv)	(a_2, inv)
(b_2, inv)	(b_1, inv)

Table 2: Two examples of possible matching assignments for the problem instance in Figure 1.

from (and including) the arrival time until (and including) the departure time of the corresponding matching. This is a conservative approach as an arrival will occupy its assigned depot track some time after arriving, and a departure will release the track allocation some time before departing from the associated station.

An example of a problem instance is given in Table 1, where a list arriving and departing events is specified. Each event occurs at a specific time and specifies the type of the units that is arriving, or the type that needs to depart. The lengths of the unit types are: $a : 200$, $b : 100$ and $c : 150$. The arrivals have to be matched to a departure or a stay in the depot (denoted by *inv*). Two matching examples are given in Table 2. Assume that we have two depot tracks available: track 1 of length 550 and track 2 of length 200. There is only one feasible parking possible using the track 1 and 2. This parking is shown in Figure 1. The matching to the left in Table 2 (Matching 1) is infeasible, because unit b_2 is blocking the departure of unit b_1. The other matching (Matching 2) is, however, feasible as no units are blocking any departure.

The shunting movements at different stations in the railway network are completely independent of each other. Hence, the respective TUSPs can be solved separately for every station. No maintenance operations are considered in our problem. Thus, train units of the same type are completely

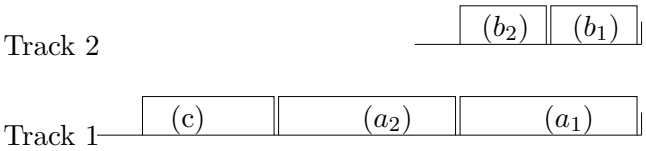

Figure 1: An example of units in Table 1 assigned to two depot tracks.

interchangeable. Internal shunting of train units is also not considered; once a train unit is parked it can only be moved when retrieved for departure. In practice, internal shunting can be performed; albeit, at the cost of using shunting personnel resources. A plan without additional movements is preferred, if possible. Finally, a certain buffer period between consecutive arrivals and departures may be desired for the same train unit. We assume that a unit must be parked at a track for at least β minutes. The value of β is parametric. A high value of β will result in a shunting plan that is more robust to delays; however, it also reduces the combinatorial solution space. In the computational experiments we set β equal to one minute.

Most of the considered problem instances have an initial inventory of units at the start of the planning horizon. Each depot track may therefore contain a number of units in a specific order. The proposed approaches adhere to this initial ordering of the units. It is also possible for the presented approaches to determine the initial parking order of the units on the track. Not having an initial ordering is a less restrictive problem, potentially giving a greater number of feasible solutions. This variant will not be pursued further in this paper, but we note that it may be relevant in a strategic or tactical planning phase.

In contrast to other work in the literature, no cost structure is defined. The TUSP is considered a feasibility problem as this is the most prominent question to answer. The presented solution approaches can be used as part of a larger framework to determine whether or not a feasible solution exists before finding the most preferred one. This is highly applicable in an operational setting where time is limited and it is crucial to detect feasibility quickly. We note, however, that almost all of the presented approaches can, without much difficulty, be extended to include an objective.

Platform Parking

Rolling stock units can be, and are in certain situations, parked on passenger platform tracks in practice. During the day passengers board and alight from trains at platforms. Consequently, units should not be parked there during the day. However, during the night there is no, or limited, traffic. If units parked on platform tracks overnight can service the first train service of the following day, then no additional shunting is required.

According to some railway operators, e.g. Nederlandse Spoorwegen (NS) and Danish State Railways (DSB), trains are, under certain circumstances, parked on platforms. The rolling stock activities for the following day are known, and it can be practical to park train units on a platform overnight if the parked composition is to depart early the next day. Depending on the track layout and the number of platforms at the station, a number of platform tracks may be eligible to be used in this way. A number of platforms must, however, be reserved in order to allow night trains or maintenance crews to operate.

At any station we assume that a certain number of platform tracks, N, can be used for overnight parking. In order to ensure a smooth operation, the first N departing train services (of the the following day) dictate which units can be parked on the N available platform tracks. A departing train service may consist of multiple units, allowing more than a single unit to be parked on a platform track. For instance, if $N = 2$, and the first two departing trains have the compositions aa and bc, then two units of type a can be assigned to the first platform (even coming from different arrival services), and a single unit of each type b and c on the second platform. In the considered variation of the problem, the extension can be handled by adding the N platform tracks as normal shunting tracks with additional restrictions. More specifically, in addition to the existing track constraints, an arrival and departure matching can only be assigned to the platform track if the arrival takes place in an appropriate time window (close to the end of the daily operation) and if the departure corresponds to one of the first train services of the following day.

3.1 General notation

In the TUSP, a set of incoming and outgoing events occurring at a shunting yard are assumed to be given. This set of events is denoted by E. Each event $e \in E$ is either an arrival or a departure. The sets E^{arr} and E^{dep} respectively contain all arrivals and departures, and together define a partition of all events; that is, $E^{arr} \cup E^{dep} = E \wedge E^{arr} \cap E^{dep} = \emptyset$. An arrival corresponds to a unit that is uncoupled at the station and must be parked, while a departure is a unit that is to be coupled at the station and must be retrieved from the shunting yard. Furthermore, we assume a set, M, of the rolling stock types and that a set, S, of shunting yard tracks are available in the depot. Two subsets of $S_d \cup S_p = S$ denote the set of depot tracks and the set of platform tracks. All definitions are summarized in Table 3.

The time at which event $e \in E$ takes place is denoted by t_e, and m_e denotes the train unit type of the corresponding event. We let c_s denote the parking capacity of track $s \in S$. This capacity is equal to the maximum length that can be stored simultaneously on track $s \in S$. The length of a train unit type $m \in M$ is assumed to be l_m. Table 4 summarises this

Set	Definition
E	Set of all events
$E^{arr} \subset E$	Set of arrival events
$E^{dep} \subset E$	Set of departing events
M	Set of train unit types
S	Set of tracks
$S_d \subset S$	Set of tracks in the depot for parking
$S_p \subset S$	Set of platform tracks

Table 3: List of defined sets

Notation	Description
t_e	Time event $e \in E$ takes place
m_e	Rolling stock type corresponding to event $e \in E$
l_m	Length of rolling stock type $m \in M$
c_s	Length of track $s \in S$

Table 4: List of defined shorthand notations

notation.

Complexity

The complexity of matching and parking trains has been addressed by multiple authors in the literature. Multiple variants of the problem exist, which are known to be $\mathcal{NP}$-hard. The shunting problem considered by Freling et al. [1] is essentially a specialization of the considered TUSP. They prove the variant to be $\mathcal{NP}$-hard by reduction from the Tram Dispatching Problem studied by Winter et al. [6], which in turn is $\mathcal{NP}$-hard.

We present a simple and informal proof, that shows that the considered TUSP is $\mathcal{NP}$-hard by reduction from the Graph Coloring Problem (GCP). In the GCP a color must be assigned to each vertex such that no adjacent vertices have the same color. Two vertices are adjacent if an edge connects them. The problem is then to decide the fewest number of colors needed. The corresponding decision problem is to decide whether a graph can be colored using k colors. The GCP is known to be $\mathcal{NP}$-hard [7].

First, the TUSP is in $\mathcal{NP}$ since a feasible solution can be verified in polynomial time. The matching constraints are verified by counting the number of assignments, the capacity constrains can be verified by looping through the events (ordered by time), and a pair-wise comparison of all matching assignments to tracks can verify that no ordering conflicts exist. Second, we argue that the TUSP is a generalization of the GCP. Given a instance of the GCP, the number of colors corresponds to the number of available tracks. The length of the tracks is set sufficiently high, such that it is never binding. The constructed TUSP instance is generated such that

only one valid matching exists by assigning a unique unit type to all train units. A vertex corresponds to a track assignment (of a matching), and the selected track represents the selected color. The edges of the graph are now used to generate a relative ordering of the arrival and departure events such that two matchings are in conflict if and only if an edge exists between the corresponding vertices in the graph. If the constructed TUSP instances contains a feasible solution, then a feasible assignment of k colors is given by the track assignments of the matchings of the TUSP instance.

4 Infeasibility Checks

A select number of efficient feasibility checks can be performed independently of any solution method. The problem instances considered in the benchmark testing of Section 6 have all passed the checks discussed in this section. There is no reason to consider an instance, which violates any of the following checks as it is inherently infeasible.

Aggregated Track Capacity

At any given time the sum of all depot track lengths must be no less than the sum of all train units that need to be parked. This aggregated constraint must hold since no feasible solution can exist if it is violated. This property is easily checked in polynomial time. Rolling stock schedules implicitly satisfy this constraint if depot capacity is modeled in the rolling stock problem.

Individual Track Capacity

The depot must have at least one feasible initial parking. At the start of the planning horizon a feasible parking must exist, otherwise no solution can exist to the TUSP either. With a given set of initial units, a feasible solution can be found by solving a Multiple Knapsack Problem (MKP). Every train unit corresponds to an item, where the capacity consumption is equal to the physical length of the unit. Each depot track corresponds to a knapsack, where the capacity is equal to the track length. Efficient algorithms for solving the MKP exist (see e.g. Pisinger [8]), making this check efficient; especially since the resulting MKP problem size is small.

In this paper, we assume that the initial parking given is feasible with respect to the mentioned knapsack constraints. However, the same constraints must be satisfied during the whole planning period, and not only for the initial parking. The same check is applied every time a unit arrives to the depot. Note that the individual track capacity can be violated even if the aggregated track capacity is satisfied.

Set	Definition
$\mathcal{A}$	Set of all matchings
$\mathcal{A}_e^{arr}$	Set of matchings where event $e \in E^{arr}$ is the arrival
$\mathcal{A}_e^{dep}$	Set of matchings where event $e \in E^{dep}$ is the departure

Table 5: List of MIP specific sets

Feasible Matching

In the common case, multiple feasible matching exists for the same problem instance, especially when ignoring depot track capacities. The reason for integrating matching and parking in the TUSP is the fact that all feasible matchings do not necessarily have a feasible parking assignment. However, if no feasible matching exists then the TUSP is infeasible as well. Detecting whether a feasible matching exists is equivalent to solving the Assignment Problem (AP) (Munkres [9]), which is solvable in polynomial time as the resulting Linear Program (LP) is totally unimodular. As the number of arrival and departure events are equal in size, the problem is the linear AP; this can be solved using specialized polynomial time algorithms, such as the Hungarian algorithm (Kuhn [10]).

5 Solution methods

In this Section we describe the different solution methods for solving the TUSP. First, the Reference MIP Method (RMM) is described in Section 5.1. In Section 5.2 the Constraint Programming Method (CPM) is presented. A column generation method and the Two-Stage Method (TSM) are presented in Section 5.3. Finally in Section 5.5 we introduce the Randomized Greedy Construction Heuristic (RGCH).

5.1 The Reference MIP Method

Arrivals and departures are linked using matchings. The matching of an arrival and a departure event is allowed if and only if sufficient time separates the events and the unit types are compatible. The set of all possible matchings is denoted by $\mathcal{A}$:

$$\mathcal{A} = \{ \quad (e, f) \quad | \quad t_e + \beta \leq t_f, \quad m_e = m_f, \quad e \in E^{arr}, \quad f \in E^{dep} \quad \}$$

The set of matchings where event $e_1 \in E^{arr}$ is the arrival is denoted by $\mathcal{A}_{e_1}^{arr}$ and the set of matchings where event $e_2 \in E^{dep}$ is the departure is denoted by $\mathcal{A}_{e_2}^{dep}$. These additional sets are summarized in Table 5.

The mathematical model contains one family of binary decision variables $X_{a,s}$. The variable $X_{a,s}$ takes a value of 1 if and only if matching $a \in \mathcal{A}$

is selected and parked on track $s \in S_d$. We extend the model to include platform parking later.

A number of constraints need to be satisfied in order to achieve a feasible solution. First, each arrival must be assigned to exactly one departure, c.f. Constraints (1). Similarly, each departure must be assigned to exactly one arrival, c.f. Constraints (2).

$$\sum_{s \in S_d} \sum_{a \in \mathcal{A}_e^{arr}} X_{a,s} = 1 \qquad \forall e \in E^{arr} \tag{1}$$

$$\sum_{s \in S_d} \sum_{a \in \mathcal{A}_f^{dep}} X_{a,s} = 1 \qquad \forall f \in E^{dep} \tag{2}$$

Constraints (1)-(2) ensure a feasible matching. The capacity of a depot track can not be exceeded at any point in time. It is therefore sufficient to ensure that the track capacity is not exceeded at every arrival, c.f. (3).

$$\begin{aligned} &\sum_{\{e' \in E^{arr} | t_{e'} \leq t_e\}} \sum_{a \in \mathcal{A}_{e'}^{arr}} X_{a,s} \cdot l_{m_{e'}} \\ &- \sum_{\{e' \in E^{dep} | t_{e'} \leq t_e\}} \sum_{a \in \mathcal{A}_{e'}^{dep}} X_{a,s} \cdot l_{m_{e'}} \leq c_s \qquad \forall e \in E^{arr}, s \in S_d \end{aligned} \tag{3}$$

For each arrival, Constraints (3) sum the contribution of past events and ensure that the used track length is less than the available track length; arrivals consume capacity, while departures release capacity.

The depot tracks are subject to LIFO restrictions. Only the out-most (top of stack) can be retrieved at any point in time. We model these restrictions by adding one constraint per pair-wise conflict to disallow such assignments, c.f. Constraints (4). It states that any two pairs of matchings cannot be assigned simultaneously if they block each others' movements.

$$X_{a,s} + X_{a',s} \leq 1 \qquad \forall s \in S_d, (a, a') \in \mathcal{C} \tag{4}$$

where

$$\begin{aligned} \mathcal{C} = \{ \quad (a, a') \quad | \quad & a = (e, f) \in \mathcal{A}, \\ & a' = (e', f') \in \mathcal{A}, \\ & t_{e'} < t_e \ \wedge \ t_{f'} < t_f \ \wedge \ t_{f'} > t_e \quad \} \end{aligned}$$

Note, that the conflict set is not track-dependent. All pair-wise conflicts are therefore repeated for every track, effectively generating very many constraints. We note that the number of constraints in the model can be reduced, possibly drastically, by replacing the pair-wise conflicts with conflict cliques, see Kroon et al [2]. In general, the problem of finding such

cliques is $\mathcal{NP}$-hard (Karp [11]). However, a number of cliques can be found heuristically in order to make the problem more tractable.

The initial inventory is not modeled directly in the above formulation. Recall, that initially parked units are modeled using arrivals and that parking (at the end of the planning horizon) is modeled using departures. Any initial unit to track assignment can be modeled by fixing the variables corresponding to the first set of events. Any order of units on depot track can also be achieved by shifting the artificial arrival times (without exceeding the real first event) to reflect the same ordering. Likewise, any final parking can be achieved using a similar modification for the artificial departures.

In the computational experiments of Section 6, a parking order is imposed. We note that in the implementation of the model all fixed variables are removed in order to obtain a more compact model.

Due to the large number of (conflict) constraints present in the model, we also introduce the Delayed Constraint MIP Method (DCMM), where these constraints are generated on-the-fly. The DCMM is solved as a MIP model, where violated conflict constraint are added as they become violated by the optimal LP solutions. Initially, no conflict constraints are added. The success of this approach depends on the fact that most conflicts will never be violated in the B&B approach of a MIP solver.

Platform Parking Extension

A given set of platforms $s \in S_p$ is available for parking arrival trips overnight. The set of decision variables is extended to include platform parking. The model now contains one variable $X_{a,s}$ for every activity $a \in \mathcal{A}$ and track $s \in S = S_d \cup S_p$. We denote the number of slots at track $s \in S_p$ for units of type $m \in M$ to be $p_{m,s}$. Constraints (5) ensure that the number of parked units does not exceed the number of units that can be parked on the platform.

$$\sum_{a \in \mathcal{A}_e^{arr}} X_{a,s} \leq p_{m,s} \qquad \forall m \in M, e \in E^{last}, s \in S_p \tag{5}$$

No LIFO ordering contains are included as the units on the track leave as a whole, and the track capacity is implicitly satisfied by construction of the $p_{m,s}$ coefficients. Preferably, only arrivals close to the end of day can be matched to the platforms. This is enforced by fixing or removing the invalid matching variables.

5.2 The Constraint Programming Method

As it is primarily a feasibility problem, the TUSP can be formulated using a Constraint Program (CP) approach. Our proposed formulation is inspired

Set	Definition
C	Set of possible compositions
Q	Set of possible composition changes
$Q_{e,s}$	Set of composition changes that are allowed after event $e \in E$ at track $s \in S$

Table 6: List of all additional sets required by the CP model

by the rolling stock composition model of Fioole et al [12], where it was originally used for Rolling Stock (Re)Scheduling; however, we use the idea of compositions and composition changes for the TUSP. Instead of assigning events to tracks, we assign compositions to tracks. For every time-interval one composition is assigned to every track individually. A composition consists of a number of train units in a specific order. Note that the empty composition, containing no train units, is a valid composition.

We let C be the set of all possible compositions and Q be the set of all possible composition changes. The set $Q_{e,s}$ consists of all feasible composition changes that can take place just after event $e \in E$ on track $s \in S$. For instance, if event e stipulates that a unit of type a is arriving, then only composition changes where a unit of type a appears on the top of the stack are included in $Q_{e,s}$. Additionally, composition changes where no units are appended or removed are included as all unaffected tracks remain unchanged. See Table 6 for an overview of the additional sets.

The first additional parameter required by the CP is i_s, which specifies the initial composition on track $s \in S$. Next, λ_e defines the predecessor event of event $e \in E$, i.e. the event that occurs just before e. Furthermore, for composition change $q \in Q$, we introduce the shorthand notation, $\texttt{In}[q]$ and $\texttt{Out}[q]$, which denote the index of the first and second composition in a composition change. Thus, the original composition and its successor composition. Finally, $\alpha_m[q]$ and $\beta_m[q]$ specify whether a unit of type m is appended or removed. These parameters are summarized in Table 7.

We define two families of decision variables. First, the integer variable $X_{e,s}$ specifies which composition is assigned track $s \in S$ just after event $e \in E$. The compositions are mapped to integer values, e.g., $X_{e,s} = 3$ stipulates that the *ab* composition is assigned to track s after event e. Recall, a composition $c \in C$ which is assigned to track $s \in S$ just after event $e \in E$ consists of all train units parked at that moment on track s, in order of arrival time. For instance, if the composition *abcd* is assigned track s after event e, it means that unit d was parked there first, thereafter unit c, then b, and finally train unit a.

The integer decision variable $Y_{e,s}$ represents the composition change that is performed on track $s \in S$ just after event $e \in E$. An example is the composition change from composition *aa* to *a*, where one unit is removed. Again,

Parameters	Description
i_s	The composition belonging to the start inventory at track $s \in S$
λ_e	The predecessor event of event $e \in E$
$\texttt{In}[q]$	The index of the first composition belonging to composition change $q \in Q$
$\texttt{Out}[q]$	The index of the second composition belonging to composition change $q \in Q$
$\alpha_m[q]$	Equals 1 if a unit of type $m \in M$ is appended to the composition on the track during composition change $q \in Q$
$\beta_m[q]$	Equals 1 if a unit of type $m \in M$ is removed from the composition on the track during composition change $q \in Q$

Table 7: List of all parameters

the integer values are mapped to the change from a specific composition to another specific composition.

The construction of the $Q_{e,s}$ set models the allowed compositions changes with respect to the depot track capacity and the LIFO restriction. Note that platform parking can also be modeled by construction of this set. First, composition changes that exceed the length of a track are not allowed. All composition changes that involve a transition to a composition that has a total length longer than the capacity of track s are removed from the set $Q_{e,s}$. Furthermore, restrictions with respect to the unit type of events are considered. First, if event $e \in E$ is an arrival of a unit of type a, then only composition changes where a unit of type a is appended and composition changes where no units are appended or removed are allowed. Second, the LIFO constraints further restrict the set of composition changes. If, for instance, the composition *abcd* was assigned to track s just after event λ_e, then there are only 2 allowed composition changes after a departure $e \in E$: $abcd \rightarrow abcd$, or $abcd \rightarrow bcd$. Units b, c, and d cannot depart as unit a is blocking them. Finally, if platform parking is allowed, as it is not allowed to park train units at platforms during the day, the only allowed composition change after events during the day is $empty \rightarrow empty$. For the last events of the day, however, platform parking can be considered. In such cases the platform track composition is restricted to being a subset of the composition of the train service which will first depart from the platform on the following day. This can be considered by only allowing those composition changes that involve transitions to compositions which are a subset of the specific departing train service composition for the following day.

The TUSP can be modeled with the following mathematical program,

which is used as a basis for the CPM:

$$X_{\lambda_e,s} = \texttt{In}[Y_{e,s}] \qquad \forall e \in E, s \in S \qquad (6)$$

$$X_{e,s} = \texttt{Out}[Y_{e,s}] \qquad \forall e \in E, s \in S \qquad (7)$$

$$\sum_{s \in S} \beta_m[Y_{e,s}] = 1 \qquad \forall e \in E^{dep}, m \in M : m_e = m \qquad (8)$$

$$\sum_{s \in S} \alpha_m[Y_{e,s}] = 1 \qquad \forall e \in E^{arr}, m \in M : m_e = m \qquad (9)$$

Constraints (6) state that the first composition of a chosen composition change on a track has to match the actual composition that is appointed before the composition change took place on the track. This actual composition is the composition that is assigned to the track just after the previous event, λ_e. A similar composition flow conservation constraint is used for the second composition of a chosen composition change. This composition must be equal to the actual composition that is appointed to track after the composition change took place, which equals the composition that is assigned to the track just after the event, e. This is modeled by Constraints (7).

Every departure event has a corresponding unit that has to depart from precisely one of the tracks in the depot. Consequently, exactly one composition change has to be selected where a unit of type m_e is removed; on all other tracks no shunting movements can take place. This is modeled by Constraints (8). A similar requirement holds for an arrival. The corresponding unit m_e has to be appended to precisely one track; this is handled by Constraints (9).

Finally, the start inventory is enforced by fixing the values for $X_{e,s}$ of an auxiliary source event e that occurs before the first event.

5.2.1 Solution procedure

The proposed model can be solved using a CP solver, effectively solving the TUSP by assigning compositions and composition changes to the events on the tracks. Similar to the rolling stock scheduling variant in [1], the model does not scale well. Long depot tracks and a high number of unit types result in a huge number of variables in practice. A large number of variables is needed for long depot tracks as the number of possible compositions increases drastically in such cases. Multiple, different unit types further increase the combinatorial solution space.

Preliminary results showed that small instances with two unit types are practical to solve. However, larger instances with four unit types quickly become impractical to solve primarily due to the memory footprint. As a remedy, the CPM can be adapted to a more practical heuristic method. We present a heuristic variant, the Constraint Programming Heuristic (CPMH),

that restricts the compositions on tracks to contain at most $\epsilon \geq 1$ different unit types. Note, that the contained types can change over the course of a planning horizon. At any time, the number of different types is at most ϵ. In effect, fewer tracks will be mixed, i.e., contain more than one unit type, in the solution. Keeping tracks homogeneous is beneficial as it also reduces the potential LIFO conflicts. The main benefit is the drastic reduction of variables in the CP model. If the heuristic finds a solution, then clearly it is feasible for the TUSP. However, if it fails to find a solution we cannot conclude that the solution is infeasible, unless ϵ is equal to the actual number of unit types.

The minimal value of ϵ that produces a feasible solution is initially unknown, therefore we begin with $\epsilon = 1$. The strategy is to solve the model using increasing values of ϵ. A time limit of γ minutes is set in every iteration. Otherwise, too much time is potentially spent searching an infeasible solution space. In this paper we divide the available time, T, uniformly by the number of unit types in the problem instance, $\gamma = {}^{T}/_{|M|}$. In every step when no solution has been found, we increase ϵ with 1 and try again, until $\epsilon = |M|$.

5.3 The Column Generation Method

To combat the potentially large number of constraints in the MIP formulation, a column generation approach can be used to solve the TUSP. In this approach the problem is decomposed by track, where each track is assigned to a set of possible matchings, termed a *matching pattern*. A matching pattern is a subset of matchings that can be feasibly parked on a given track over the planning horizon. In particular, it is a set of matchings that satisfies the LIFO requirements as well as the available track length restriction. A large number of possible matching patterns exist thus the approach relies on dynamic generation of variables that represent promising matching patterns. In this paper, the solution method is referred to as the Column Generation Method (CGM).

The proposed formulation is based on the methodology presented by Freling et al. [1], with the exception that in our work the matching and parking problems are not solved separately. We present a model and solution framework that simultaneously solves both problems. To assist in the description of the model, we introduce the set $\mathcal{P}_s$, which denotes the set of all feasible matching patterns for track $s \in S$. Note, that platform parking and LIFO constraints are already satisfied in this set. A binary decision variable $X_{p,s}$ is defined for each $s \in S$ and $p \in \mathcal{P}_s$ and governs the inclusion of the corresponding matching pattern in the final solution. A value of one indicates that the matching pattern is chosen, while a value of zero indicates otherwise. As the majority of the constraints are embedded in the column construction phase, the problem can be formulated as a large generalized

set partitioning problem. In the model, at most one matching pattern is assigned to each track. Further, each arrival and departure must appear in at most one matching pattern, otherwise it is left uncovered. Binary variables Y_e, where $e \in E^{arr}$, and Z_e, where $e \in E^{dep}$, are used to indicate whether an arrival, respectively departure, is matched or not. These variables are penalized in the objective function by, Γ, thus providing an incentive for the model to match as many events as possible. Finally, the binary parameter $\alpha_{e,p}$ indicates whether or not event $e \in E$ is contained in matching pattern $p \in \mathcal{P}_s$. The full binary integer program is given as follows.

$$\text{Minimize:} \quad \sum_{e \in E^{arr}} \Gamma Y_e + \sum_{e \in E^{dep}} \Gamma Z_e \tag{10}$$

subject to:

$$\sum_{p \in \mathcal{P}_s} X_{p,s} \leq 1 \qquad \forall s \in S, \ (\boldsymbol{\mu}) \tag{11}$$

$$\sum_{s \in S} \sum_{p \in \mathcal{P}_s} \alpha_{e,p} X_{p,s} + Y_e = 1 \qquad \forall e \in E^{arr}, \ (\boldsymbol{\pi}) \tag{12}$$

$$\sum_{s \in S} \sum_{a \in \mathcal{P}_s} \alpha_{e,p} X_{p,s} + Z_e = 1 \qquad \forall e \in E^{dep}, \ (\boldsymbol{\gamma}) \tag{13}$$

$$X_{p,s} \in \{0,1\} \qquad \forall s \in S, p \in \mathcal{P}_s, \tag{14}$$

$$Y_e \in \{0,1\} \qquad \forall e \in E^{arr}, \tag{15}$$

$$Z_e \in \{0,1\} \qquad \forall e \in E^{dep}. \tag{16}$$

The objective, given in (10), minimizes the penalties incurred from uncovered events. Constraints (11) ensure each depot track is assigned at most one matching pattern. Constraint sets (12) and (13) enforce the requirement that each arrival and departure appears in one of the selected matching patterns, or is left uncovered. Finally, variable domains are specified by constraints (14)-(16). We refer to Model (10)-(16) as the *master* problem.

The Master Problem

Given the exponential number of matching patterns in any real-life example it is impractical to enumerate all corresponding columns and solve this formulation. In our solution method, a subset (restricted set) of the possible matching patterns are included. We relax the integrality restrictions and associated bounds given by (14)-(16). A *relaxed, restricted master problem* (RRMP) is obtained. Using the optimal dual solution vector $(\boldsymbol{\mu}^*, \boldsymbol{\pi}^*, \boldsymbol{\gamma}^*)$ to this relaxed problem, a *pricing* problem is solved to determine if any favourable matching patterns exist. Promising variables are inserted iteratively into the restricted master problem until none exist - implying that the

LP solution is proven optimal. By iterating between the RRMP and several pricing problems (typically one for each track), one can limit the search for the optimal solution to model (10)-(16) to include only those matching patterns that have the potential to improve the objective value. For a general introduction to column generation the reader is referred to [13].

The Pricing Problem

The pricing problem requires one to find a favourable set of matchings that can feasibly be parked on a given track. In other words, given an optimal solution to the RRMP, one must solve up to $|S_d| + |S_p|$ pricing problems at any column generation iteration to determine if any improving matching pattern exists. To find such patterns we present an approach that finds shortest paths in a directed *pricing* graph.

In the *pricing* graph there is one node for every possible matching, one node for every arrival (corresponding to not parking the arrival), and a source and sink node. The graph is layered by matchings for each arrival (including the node corresponding to not parking the arrival) and these layers are ordered in increasing arrival time. Arcs connect matchings in one layer with those of the subsequent layer - providing the two matchings can feasibly use the same track. The source node is connected to each matching in the first layer, while each matching in the last layer is connected to the sink. There is a cost on any arc entering a matching node equal to dual contribution to the reduced cost of the arrival and departure matched in the matching. E.g. if events $e \in E^{arr}$ and $e' \in E^{dep}$ are matched, the cost on any arc entering the node corresponding to this matching will have a cost of $-(\pi_e + \gamma_{e'})$. An example of such a network is given in Figure 2

As we must observe the available track length and satisfy the LIFO requirement when generating matching patterns, a resource constrained shortest path problem must be solved. Consequently, a standard label setting algorithm is used to identify paths in this network. The algorithm is similar to that of Freling et al. [1]; however, as we must also simultaneously find the matching, the proposed network is much bigger. Additionally, we must also keep track of previously matched departures. The ordering of the layers ensures that each arrival is matched exactly once (or not parked); however, a path in the presented graph can match the same departure more than once. This is indeed likely if doing it improves the reduced cost resulting in matching patterns that cover the same departure multiple times. In addition to being infeasible, these patterns also dominate feasible labels. For an exact approach, the only option would be to weaken the dominance (ensuring all possibilities) are generated; however, this is impractical with large networks. In this paper, we adopt the second approach; i.e. a label can not visit a matching if the departure associated with the matching has been previously matched. This is heuristic; however, it ensures the solution

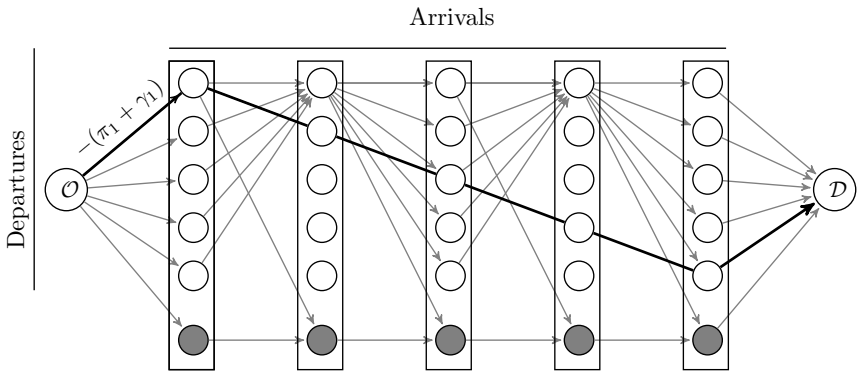

Figure 2: An example subproblem network with five arrivals and five departures. Every node corresponds to either the matching of an arrival and a departure or an unmatched arrival (gray nodes). All paths originating at the source ($\mathcal{O}$) and terminating at the sink ($\mathcal{D}$) represent a matching pattern, which is feasible if the resource constraints are respected. An example of such a path is given in black. Note that not all arcs and costs are shown.

times of the pricing problem are manageable.

To ensure exactness of the column generation, the heuristic column generation approach outlined can be complemented with a MIP solve. For instance, one can resort to a MIP when the column generation fails to identify a negative reduced column. This MIP, however, would be similar in structure to that described in Section 5.1, with the exception that only the "best" set of matchings need to be decided for the shunting track in question. For large problems, this is expected to be slow.

5.4 The Two-Stage Method

Given a matching to an instance of the TUSP, the remaining problem simply entails parking the set of matchings. A problem instance may contain multiple feasible matchings for which a feasible parking exists. Solving these two problems in isolation is expected to be easier than solving them jointly. This motivates the Two-Stage Method (TSM), where in the first stage a feasible matching of arriving and departing units is generated, while in the second stage the method tries to park the found matchings.

The matching and the parking problem can be solved using different methods. For the matching problem we resort to a MIP approach for two reasons. First, it is easy to formulate and implement a MIP for the matching problem. Second, as mentioned in Section 4, the resulting LP is totally unimodular. We also adopt a MIP approach for solving the parking problem. Freling et al. [1] proposed a column generation approach for solving this

problem; however, the fast runtimes of our approach gave us no reason to pursue a more complicated framework.

For the matching problem formulation a binary decision variable X_a is introduced that indicates whether a given matching $a \in \mathcal{A}$ is selected or not. In a feasible matching, each arrival and departure should appear in exactly one matching. The resulting constraints of the MIP are given below.

$$\sum_{a \in \mathcal{A}_e^{arr}} X_a = 1 \qquad \forall e \in E^{arr}, \tag{17}$$

$$\sum_{a \in \mathcal{A}_e^{dep}} X_a = 1 \qquad \forall e \in E^{dep}, \tag{18}$$

$$X_a \in \{0, 1\} \qquad \forall a \in \mathcal{A}. \tag{19}$$

There is no objective used in the MIP, since we are interested in feasibility only. Constraints (17) and (18) ensure, respectively, that each arrival and departure event appears in exactly one matching. The variable domains are given by (19). Several solutions, i.e. matchings, to the model may exist and it may therefore be useful to guide the solution in a more advanced approach.

If a solution to the matching problem problem exists, we proceed to the second stage and attempt to park them. For this, we use the MIP described by Haahr et al. [14], which is identical in structure to the reference MIP approach described earlier.

The TSM is similar to what is described by Freling et al. [1]. However, the matching problem we propose is slightly different and we use a MIP solver for the parking problem. A heuristic column generation framework is described in Freling et al. [1].

5.5 The Randomized Greedy Construction Heuristic

Modeling ordering constraints efficiently in integer linear programs such as LIFO constraints is cumbersome. Our proposed solution methods overcome this modelling issue by either adding all pairwise conflicts, enumerating all possible transition states, or by generating feasible parking patterns. All methods have scaling issues, by the number of constraints or variables. In contrast, modelling one or multiple stacks programmatically is fairly straightforward.

We propose a heuristic that greedily assigns arrivals and departures to and from tracks. The important key ingredients are the efficiency of the construction and the randomization of the greedy choice. Together these characteristics allow the heuristic method to try multiple paths of track assignments and extractions within a short time. The method terminates with the first feasible solution.

Algorithm 1 Randomize Greedy Construction Heuristic

```
1: Input: Track set S
2: Input: Event set E, ordered by time
3: Output: Matching set M
4: M ← ∅
5: S ← InitializeEmptyTrackStacks()
6: for e ∈ E do
7:     if Type(e) =Arrival then
8:         s ← FindRandomCompatibleTrack(S)
9:         if s = ∅ then
10:            return ∅
11:        else
12:            S[s] ←Push(S[s], e)
13:    else
14:        s ← FindRandomCompatibleUnitType(S)
15:        if s = ∅ then
16:            return ∅
17:        else
18:            (S[s], e′) ←Pop(S[s])
19:            M ← M ∪ {(e, e′, s)}
```

An overview of the heuristic is shown in Algorithm 1. The input to the heuristic is the set of events to process and the set of available tracks. The main loop processes events by ascending time. In case of an arrival, a random compatible track is sought, i.e., any track that has sufficient remaining length to hold the arriving train unit. Many candidates may exist, thus the following selection criteria are used:

1. A track where the existing outmost unit has the same type
2. A track which is empty
3. Any track with sufficient capacity

The goal is to group the same type of units, and avoid stacking different unit types on the same tracks. Note, units of the same type do not block each other as they are interchangeable. In order to avoid a standstill in certain situations at depots with scarce capacity, the first and second criteria are skipped with a low probability. The situation occurs when unit types must be mixed on tracks in order to utilize the capacity fully.

In case of a departure, tracks are processed in a random order. The track with the correct unit type (on the top of the stack) is selected. Here it might be worthwhile considering a selection criteria approach based on the effect of removing this unit. However, preliminary results show that this simple extraction rule is sufficiently effective.

The algorithm output is a list, where every element defines an arrival, a departure matching, and a specific track. On arrival the unit is parked on the track, and the specified departure extracts the unit from the same track.

The heuristic is able to evaluate one construction path very quickly, thus it is embedded in an iterative loop, where the heuristic is applied with different seeds to initialize the random number generator. Every iteration thus essentially restarts the whole process. The loop continues until either a feasible solution is found or the time limit is reached.

5.6 Type and Track Decomposition

Some problem instances contain many events, unit types, or long tracks. This results in a large number of possible matchings or track assignments, which makes the problem impractical to solve using exact methods. For example, the CPM and RMM require too much memory to represent the mathematical formulas as they contain an explicit representation of the problem.

The solution space can be reduced significantly by decomposing the problem instances by unit types and tracks. In the proposed decomposition, a unit type is restricted to park on a select subset of tracks. The partitioning of the tracks and unit types can be performed such that the original problem decomposes into several independent problems, which can be solved individually in sequence or in parallel. We consider such partitions where both the unit types and tracks are partitioned into K groups, such that one group of unit types is assigned to one group of tracks. By construction, no interaction needs to takes place across the selected groups.

The decomposition divides the problem into a number of smaller independent subproblems. The primary advantage is that solving all resulting subproblems is easier than solving the full original problem. A second advantage is that the decomposition is independent of the underlying solver. The subproblems can be solved using any solution method for the TUSP. The primary disadvantage is that the resulting framework is inherently heuristic as the decomposition restricts the original solution space. Feasible solutions found using the decompositions are naturally also feasible in the original problem; however, we cannot conclude that a problem instance is infeasible if any one subproblem is infeasible. Another drawback of the proposed decomposition is the existence of multiple partitions. Some of the partitions may contain feasible solutions to all subproblems while others may not. Determining the partition is therefore another problem that must be addressed.

Due to the scope of this paper, we only propose a simple method of finding eligible partitions that will be tested in Section 6. For the selected problem instances we generated a number of random partitions. Partitions are rejected if they do not pass the checks described in Section 4.

6 Computational Results

The presented solution methods are benchmarked on different classes of instances, which originate from three different railway networks in two different countries. Four classes, summarized in Tables 8, 9 and 10, are considered: STOG, DSB, NS, and NS-HARD. These instances are based on the railway networks of the Danish State Railways (DSB) and the Nederlandse Spoorwegen (NS) - the principal operators in Denmark and the Netherlands respectively.

All instances, except the DSB class, have been generated using a rolling stock optimizer. The events going in and out of the depots are extracted from the optimized schedule and define separate instances for each depot. Information about fleet size, train unit types, and depot track lengths are given by the railway operators.

The STOG class consists of twelve distinct rolling stock schedules obtained by optimizing the suburban railway network in the greater Copenhagen area (DSB S-tog). This gives up to twelve different event lists per station. Identical problem instances have been eliminated resulting in a total of 96 instances for the STOG class.

The DSB class consists of real-life data for a recurring weekly schedule at the busiest station in Denmark, which is located in the center of Copenhagen. Every day in the weekly schedule is unique thus resulting in seven instances for the DSB class.

The NS class consists of ten distinct rolling stock schedules for the whole country. This leads to ten different problem instances at eleven different stations. There are large differences between the event lists per station; some are large and some are small. Consequently, there are both difficult and relatively simple problem instances for the NS class.

The NS-HARD class is identical to the NS class, expect that fewer tracks are now available at busy stations. These artificial cases are therefore constrained in terms of capacity, in turn reducing the number of feasible parking plans. These have been included as an attempt to stress test the solution methods.

All computation experiments are performed on a dedicated machine equipped with two Intel(R) Xeon(R) CPU X5550 (2.67GHz) processors and 24 gigabytes of main memory. Version 12.6 of the commercial solver CPLEX is used to solve the MIP and CP based approaches. A time limit of 900 seconds is set for all experiments.

The following is a short summary of the solution methods proposed in Section 5 benchmarked in this section.

RMM A reference MIP approach solved using the CPLEX MIP solver.

DCMM A variant of the RMM where the pairwise order conflict constraints are generated *on-the-fly*.

Class	Depot	Min	Max	Tracks	Length	Types
STOG	BA	12	14	4	936	2
STOG	FM	16	66	4	727	2
STOG	FS	26	58	6	1 020	2
STOG	HI	20	54	6	1 635	2
STOG	HOT	2	22	1	173	2
STOG	HTAA	20	68	35	3 272	2
STOG	KH	36	78	9	2 753	2
STOG	KJ	24	60	6	1 115	2
STOG	KL	6	66	3	558	2
STOG	UND	24	46	6	1 670	2
DSB	KK	326	518	10	3 878	12

Table 8: Summary of instances: The first column indicates to which class the problem belongs. The second column defines the station. The third and fourth column show the minimum and maximum number of events taking place at the station. The fifth column presents the number of depot tracks available within the station and the sixth column defines the total length of all depot tracks combined. Finally, the seventh column defines the number of different rolling stock types that need to be parked within the station.

Class	Depot	Min	Max	Tracks	Length	Types
NS	AMR	157	159	9	2 267	4
NS	DDR	162	162	4	939	4
NS	EHV	153	179	20	7 061	4
NS	EKZ	97	97	5	1 590	4
NS	GVC	742	744	17	5 690	4
NS	HDR	82	82	3	1 143	4
NS	HFDO	561	561	8	3 020	4
NS	HN	75	75	12	2 023	4
NS	NM	268	268	25	6 495	4
NS	RTD	378	380	22	5 384	4
NS	ZP	87	87	9	4 127	4

Table 9: Continued summary of instances.

Class	Depot	Min	Max	Tracks	Length	Types
NS-HARD	GVC14	742	744	14	4 712	4
NS-HARD	HFDO5A	561	561	5	1 934	4
NS-HARD	HFDO5B	561	561	5	1 898	4
NS-HARD	HFDO6	561	561	6	2 197	4
NS-HARD	NM10	268	268	10	2 457	4
NS-HARD	NM11	268	268	11	2 657	4
NS-HARD	RTD11	378	380	11	3 085	4
NS-HARD	RTD12	378	380	12	3 410	4
NS-HARD	RTD13	378	380	13	3 669	4

Table 10: Continued summary of instances.

CPM A constraint program formulation inspired by the composition model in [12]. The formulation is solved using the CPLEX constraint program solver.

CPMH A variant of the CPM where the number of different unit types assigned to the same track is limited.

RGCH A randomized greedy construction heuristic that is executed multiple times with different initial seeds.

CGM A column generation approach that assigns matching patterns to tracks.

TSM A two-stage decomposition method that solves the matching and parking problem in sequence using MIP approaches.

Before presenting the results in detail, we first note that the CGM, an extension of a method proposed in literature [1], is discarded from further analysis. The performance of this method on all instances was always inferior in comparison to the other methods. For small cases the time it took to produce an optimal solution to the linear programming relaxation was significantly greater than the time it took the MIP based approaches to produce a feasible solution. For the larger instances, CGM was unable to solve the root node relaxation (in a Branch-and-Price (B&P) framework) to LP optimality within the time limit in most cases. These results are consistent with the study in [14], where a similar column generation approach was outperformed by a MIP approach for the parking problem only.

Table 11 and 13 show a comparison overview. Table 11 shows the number of instances for which a feasible solution is found per problem class per method within the time limit. Table 13 shows the average runtimes.

In the STOG class, which contains the smallest problem instances, 94 out of the 96 instances are feasible. All methods, except the TSM, were able to find the solutions. The TSM was unable to find a feasible solution for

Class	No	RMM	DCMM	CPM	CPMH	RGCH	TSM
STOG	96	94	94	94	94	94	93
DSB	7	0	7	0	0	7	7
NS	110	0	93	84	101	110	110
NS-HARD	90	0	27	70	83	70	90

Table 11: Number of feasible instances found by the methods.

Class	RMM	DCMM	CPM	CPMH	RGCH	TSM
STOG	2	2	2	2	0	0
DSB	0	0	0	0	0	0
NS	0	0	0	0	0	0
NS-HARD	0	0	0	0	0	0

Table 12: Number of instances proved to be infeasible by the methods.

one of those instances. The average solution time is less than one second for all methods.

Solving the mathematical formulations of the other classes directly proves to be impractical. We observe that the RMM fails to solve all but the relatively small STOG problem instances. Significantly more cases can be solved by using the more efficient DCMM, CPM and CPMH variants. The DCMM can solve all DSB instances and a large portion of the NS instances, but only a few of the NS-HARD instances. The time limit becomes a prohibiting factor when using the DCMM. The CPMH is more successful at solving the NS and NS-HARD classes but unable to solve the DSB class due to the large number of unit types. The CPM performs relatively well compared to the RMM, but the performance is clearly dominated by the CPMH in terms of solutions found and average runtimes.

The CPMH, RGCH and TSM perform well on all the realistic instances as they are able to find the same number of feasible solutions. Further, RGCH and TSM are able to identify a feasible solution within a few seconds in average. However, for the artificial class of problem instances TSM proves to be the most efficient heuristic. The CPMH was unable to solve some of the larger instances (GVC), while the RGCH was unable to solve the more constrained instances (HFDO5A and HFDO5B).

Table 12 shows the number of instances for which infeasibility is proven per method within the time limit. First, we note that the heuristic methods RGCH is by definition not able to proof infeasibility. The two infeasible instances in the benchmark were detected by all exact approaches and the CPMH. The TSM was unable to prove infeasibility as at least one feasible matching exists.

Class	RMM	DCMM	CPM	CPMH	RGCH	TSM
STOG	0.5	0.4	1.3	0.6	0.0	0.0
DSB		255.8			0.1	1.3
NS		148.0	20.5	6.0	0.0	5.8
NS-HARD		267.5	78.2	33.3	0.3	1.1

Table 13: Average runtimes for finding solutions grouped by method

Based on these results we can conclude that both the TSM and the RGCH method are very fast and efficient in finding feasible solutions. The CPMH is somewhat slower, but also successful in identifying feasible solutions in many cases. The RMM is clearly dominated by the DCMM, and the CPM by the CPMH. The DCMM solves fewer instances than the CPMH and requires more runtime, however it can, in contrast, solve some instances with a higher number of unit types.

Track Splitting

The problem can be decomposed into several independent problems by partitioning the full problem by unit types and tracks as described in Section 5.6. In this section we investigate whether this decomposition technique can improve the tractability of the exact methods.

We have randomly generated a number of partitions of a selected set of problem instances with the following procedure. First, a random number of groups is selected. Second, tracks and unit types are assigned randomly to the available groups. If any group has an empty set of unit types or tracks, then the whole generation is rejected. Further, it is ensured that the maximum depot capacity required by the unit types is less than the capacity of the tracks in the group. Finally, if units are positioned initially in the depot, then this naturally adds constraints to the generation of the groups.

The DCMM, CPMH and RGCH have been considered in this benchmark as they were unable to solve several instances in the previous section. The benchmark consists of large instances of the NS class that were unsolved by the RMM, DCMM and CPM. Decomposing this class of problems reduces the size of the underlying mathematical models significantly. Further, we consider two cases of the DSB class that were unsolved by the RMM, CPM and CPMH. A decomposition of these instances drastically reduces the number of variables and constraints needed by the CP model since the resulting number of different unit types is decreased.

An overview of the generated instances and results are shown in Table 14. The average runtimes are listed in Table 15. The HFDO, GVC and RTD instances were very successfully decomposed as all the resulting subproblems were feasible. The considered solution methods were able to

				DCMM			CPMH			RGCH	
Instance	#	F	Sub.	F	I	T	F	I	T	F	T
HFDO	10	10	20	20	0	0	20	0	0	20	0
GVC	10	10	20	4	0	16	20	0	0	20	0
RTD	10	10	30	30	0	0	30	0	0	30	0
DSB1	25	19	71	65	6	0	43	2	26	50	21
DSB2	25	18	71	64	6	1	40	4	27	50	21

Table 14: Summary of results achieved when running different methods on problem instances decomposed by splitting tracks and unit types. The columns respectively show the instance considered, number of decompositions, number of feasible decompositions, number of generated subproblems, and finally the number of **F**easible, **I**nfeasible and **T**imed-out instances for every method.

Instance	No	DCMM	CPMH	RGCH
HFDO	10	83.4	13.3	0.1
GVC	10	3.2	9.9	0.0
RTD	10	18.0	1.7	0.0
DSB1	25	42.2	26.4	0.1
DSB2	25	32.6	30.8	0.1

Table 15: Summary of average runtimes achieved when running different methods on problem instances decomposed by splitting tracks and unit types. The columns respectively show the instance considered, number of decompositions and finally the average runtime for all found solutions.

solve these instances efficiently, except for the DCMM which was unable to produce a feasible solution for the subproblems of the largest instance. Nevertheless, DCMM is able to solve more instances using this decomposition technique. The DSB1 and DSB2 instances were on the contrary decomposed into both feasible and infeasible subproblems. The DCMM is able to solve all subproblems, except one, efficiently. The CPMH is now able to solve more than half of the subproblems but several remained unresolved. This is an improvement compared to the non-decomposed results. Interestingly, decomposing the problem proved unhelpful for the RGCH as many feasible subproblems were left unsolved. The RGCH was able to solve the original instances of the problems. A reduction in computational time is observed for both the DCMM and the RGCH in Table 15.

Instance	Cases	DCMM	CPMH	RGCH	TSM
KH	3	2	3	3	1
FM	3	3	3	3	0
EHV	3	3	3	3	3
HDR	3	3	3	3	0

Table 16: Summary of results achieved when running different methods on problem instances with overnight parking. The columns respectively show the instance considered, number of problem instances, and the number of solved instances for every method.

Overnight Parking

In the common case all units leave the depots during the early hours and enter the depot at the end of the day. Some units enter and leave the depots during the day, e.g. before and after rush hour periods. The capacity at depots is therefore not very limiting during the day, which makes it easy to plan this intermediate period.

The considered problem instances do not stipulate any particular parking order at the end of the day. Consequently, no ordering conflicts arise regardless of the final track assignment, when the depots are close to being at capacity. Realistically, a smooth transition from one day to another is desirable, making sure that units can leave the depot in a conflict free manner the following day. In our final benchmark, we combine the planning instances to include the events for two days of operation, thus forcing the solution methods to consider the overnight parking.

Tables 16 and 17 show the results of the overnight parking instances. The instances are naturally larger than the original ones, and require more time to solve as two days of events have been combined. The results show that the considered methods, except the TSM and DCMM, can efficiently solve all instances. All methods except the TSM integrate matching and parking, where the RGCH does this by trying many different matchings in the solution construction. The TSM first finds one feasible matching and tries to park the matched events. Fixing the matching in a early stage does, however, restrict the flexibility when resolving the ordering conflicts. In contrast to the other benchmarks, these problem instances contain at least one very busy period, where the depots are close to being at capacity. Evidently, an approach that fixes, i.e., only considers one matching, may very well fail to find a feasible solution. The average runtimes shown in Table 17 reveal that the CPMH is faster than the DCMM in general when considering these extended instances. We note, however that these instances only contain 2-4 different unit types. The performance of the CPMH is expected to decrease with higher numbers of different unit types.

Instance	Cases	DCMM	CPMH	RGCH	TSM
KH	3	449.5	223.0	0.1	12.3
FM	3	8.2	1.1	0.0	0.0
EHV	3	244.4	11.9	0.1	0.4
HDR	3	17.7	1.0	0.0	0.1

Table 17: Summary of average runtimes achieved when running different methods on problem instances with overnight parking. The columns respectively show the instance considered, number of problem instances, and average runtimes.

7 Conclusion

In this paper we have developed and benchmarked different models and solution approaches to solve the Train Unit Shunting Problem. Given a feasible rolling stock circulation, the objective of the solution approaches is to find a valid matching and shunting plan. A number of computational experiments have been performed on multiple problem instances from three different railway operators. The benchmark highlights strengths and weaknesses of the considered approaches. A platform parking extension is described, as platform tracks are currently used for overnight parking in some railway operations.

The main benchmark, consisting of multiple daily problem instances, revealed the main weakness of the exact models based on mathematical models, i.e. RMM and CPM. These approaches were outperformed by the other approaches. The resulting mathematical models quickly consumed more than 24 gigabytes of memory due to the large number of constraints and/or variables required. Using delayed constraint generation (for the RMM) the DCMM is able to solve significantly more instances. The heuristic extension CPMH method (of CPM) further outperforms the DCMM when considering the instances with a small number of different unit types. In turn, the DCMM can solve the instances with a high number of unit types. On average, the solved instances were solved in a few minutes by these methods.

A randomized construction heuristic, the RGCH, was able to solve almost all instances within one second. However, some harder and artificially generated instances were left unsolved by the RGCH. In the main benchmark the non-integrated method TSM proved to be most successful, solving all but one of the feasible instances within a few seconds. Ironically, the unsolved instance was a relatively small problem instance. Finally, a column generation approach, the CGM, was also considered, however, it was not benchmarked as it was always inferior to the MIP/CP based approaches on small instances and required too much computation time to solve larger instances.

The TUSP can be decomposed by splitting the available tracks and unit types into several independent and smaller subproblems. A set of the larger instances were decomposed in a second benchmark. In general, most of the the resulting partitions were feasible. Using this decomposition the DCMM and CPMH were able to solve more problem instances than before, but not all. In fact, the RGCH performed worse than before as it was unable to find solutions for some of the problems that were solved originally.

In a final benchmark a number of instances were combined in order to solve two days of operation. Interestingly, the results show that no added difficulty is introduced when using the integrated methods. However, the TSM is now unable to solve most of the problem instances.

The considered solution approaches have both strengths and weaknesses. The results show that no method is superior. Solving the full mathematical formulations, i.e. RMM, CPM, directly proves to be ineffective. The DCMM is able to prove infeasibility in most cases. In addition, note that the proposed feasibility checks in Section 4 are very efficient - few instances were infeasible in general. Very fast solutions can be found using the RGCH but it proves to be inefficient for the more constrained instances. Finally, the TSM solves more instances using a few seconds, but has the drawback of using a fixed matching, which can lead to premature infeasibility. In conclusion, given the low runtime requirement of the RGCH and the TSM, these approaches form a reasonable choice as a first step in any solution framework. If no solution is found the DCMM and the CPMH can be adopted in a subsequent phase.

This paper focusses only on finding a feasible solution. There is no distinction made between any feasible solution. In future research it might be interesting to extend the models by considering an objective in order to find a solution with, for instance, homogeneous tracks. Furthermore, several important restrictions have not been considered. If, for instance, the rolling stock circulation passes our feasibility check, it might still be infeasible with respect to the available crew members present for shunting operations at a station. Consequently, the crew has to be taken into account in the TUSP in future research. Other practical aspects needs to be taken into account as well, e.g. parking whole compositions instead of single units, units might require maintenance at the station, and cyclic rolling stock circulations. In future research it would be interesting to include some or all of these aspects in the TUSP.

References

[1] R. Freling, R. M. Lentink, L. G. Kroon, D. Huisman, Shunting of passenger train units in a railway station, Transportation Science 39 (2) (2005) 261–272.

[2] L. G. Kroon, R. M. Lentink, A. Schrijver, Shunting of passenger train units: an integrated approach, Transportation Science 42 (4) (2008) 436–449.

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

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

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

[6] T. Winter, Online and real-time dispatching problems, Ph.D. thesis, Technical University, Braunschweig, Germany (1999).

[7] M. R. Garey, D. S. Johnson, L. Stockmeyer, Some simplified np-complete problems, in: Proceedings of the sixth annual ACM symposium on Theory of computing, ACM, 1974, pp. 47–63.

[8] D. Pisinger, Algorithms for knapsack problems, Ph.D. thesis, DIKU, University of Copenhagen, Denmark, technical Report 95-1 (1995).

[9] J. Munkres, Algorithms for the assignment and transportation problems, Journal of the Society for Industrial and Applied Mathematics 5 (1) (1957) 32–38.

[10] H. W. Kuhn, The hungarian method for the assignment problem, Naval research logistics quarterly 2 (1-2) (1955) 83–97.

[11] R. M. Karp, Reducibility among combinatorial problems, Springer, 1972.

[12] P.-J. Fioole, L. Kroon, G. Maroti, A. Schrijver, A rolling stock circulation model for combining and splitting of passenger trains, European Journal of Operational Research 174 (2) (2006) 1281–1297.

[13] M. Lubbecke, J. Desrosiers, Selected topics in column generation, Operations Research 53 (2004) 1007–1023.

[14] J. Haahr, R. Lusby, J. Larsen, D. Pisinger, Simultaneously Recovering Rolling Stock Schedules and Depot Plans Under Disruption, Proceedings of the 13th Conference on Advanced Systems in Public Transport (CASPT), 2015.

ERIM Report Series *Research in Management*	
ERIM Report Series reference number	ERS–2015–013–LIS
Date of publication	2015–10–15
Version	15–10–2015
Number of pages	35
Persistent URL for paper	http://hdl.handle.net/1765/78820
Email address corresponding author	jwagenaar@rsm.nl
Address	Erasmus Research Institute of Management (ERIM) RSM Erasmus University / Erasmus School of Economics Erasmus University Rotterdam PO Box 1738 3000 DR Rotterdam, The Netherlands Phone: +31104081182 Fax: +31104089640 Email: info@erim.eur.nl Internet: http://www.erim.eur.nl
Availability	The ERIM Report Series is distributed through the following platforms: RePub, the EUR institutional repository Social Science Research Network (SSRN) Research Papers in Economics (RePEc)
Classifications	The electronic versions of the papers in the ERIM Report Series contain bibliographic metadata from the following classification systems: Library of Congress Classification (LCC) Journal of Economic Literature (JEL) ACM Computing Classification System Inspec Classification Scheme (ICS)