Optimization methods for the Train Unit Shunting Problem

Haahr, Jørgen Thorlund; Lusby, Richard Martin ; Wagenaar, Joris Camiel

Published in:
European Journal of Operational Research

Link to article, DOI:
10.1016/j.ejor.2017.03.068

Publication date:
2017

Document Version
Peer reviewed version

Citation (APA):
Haahr, J. T., Lusby, R. M., & Wagenaar, J. C. (2017). Optimization methods for the Train Unit Shunting Problem. *European Journal of Operational Research*, *262*(3), 981-995. https://doi.org/10.1016/j.ejor.2017.03.068

Optimization Methods for the Train Unit Shunting Problem

Jørgen Thorlund Haahr[1], Richard M. Lusby[1], and Joris Camiel 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 6, 2017

Abstract

We consider the Train Unit Shunting Problem, an important planning problem for passenger railway operators. This problem entails assigning train units from shunting yards to scheduled train services in such a way that the resulting operations are without conflicts. The problem 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 an extensive comparison benchmark of multiple solution approaches for this problem, some of which are novel. In particular, we develop a constraint programming formulation, a column generation approach, and a randomized greedy heuristic. We compare and benchmark these approaches with two existing methods, 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: *Transportation, Passenger Railway Optimization, Shunting, Matching, Parking*

1 Introduction

Passenger railways are an important mode of transportation in many countries. Travelers depend on a safe, reliable and timely service. For a rail operator, providing this service requires the careful planning of trains, personnel and infrastructure. Planning problems that must be addressed include, but are not limited to the following: *i)* determining a timetable, which stipulates the arrival and departure times of the train services to operate; *ii)* creating a rolling stock schedule, specifying a feasible fleet circulation by assigning train units to timetabled train services; and *iii)* constructing a crew plan by assigning certified train personnel to operate the trains. Due to the complexity of each of the underlying optimization problems, the planning problems are usually solved sequentially and in isolation.

In this paper, we present a comparison of optimization methods for determining whether a scheduled set of *shunting movements* at a given depot is feasible with respect to the layout of the depot. A depot (or shunting yard) is a storage facility that is usually located in the close vicinity of a railway station. It typically consists of several parallel tracks on which train units not in service can be parked. A shunting movement refers to the process of driving a unit to (or from) a depot track from (or to) a platform in the station and is induced whenever the train *composition* changes on successive train services. Multiple units can be assigned to the same train service (i.e. coupled in a convoy), and this ordered collection of units is termed a composition. Compositions are typically designed to meet the forecast passenger demand, while not using more train units than is necessary. Composition changes occur when, for instance, a train unit is either 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 is parked and awaits its next service. In the second case a train unit must be retrieved from the depot and coupled to a train service.

The rolling stock scheduling problem determines which train unit *types* to assign to each timetabled train service. Two train unit types typically differ in their respective physical characteristics, e.g. length and passenger capacity. We assume train units of the same type to be interchangeable. Shunting movements are usually not considered when planning the rolling stock schedule. It is often assumed that these can be resolved in a later phase. The assumption is that the capacity as well as the infrastructure layout of any depot on the network is sufficient to cater for the necessary shunting movements implied by the rolling stock schedule. However, for railway networks where depot capacity is scarce, it may not always be possible to feasibly perform the resulting shunting movements. Furthermore, the planning of shunting movements is not a trivial problem. This is especially true at larger stations where many shunting movements may occur over the course of a day and where there can be a number of depot tracks of different lengths. Effective methods for finding feasible solutions to, or proving the infeasibility of, the so-called Train Unit Shunting Problem (TUSP) are therefore essential.

A railway network normally contains several depots, and the set of shunting movements at each of them can be deduced from a solution to the rolling stock scheduling problem. Scheduling the rolling stock is beyond the scope of this paper, and we assume a solution to this problem is available as input to the TUSP. From the given rolling stock schedule all arrivals at and departures from depots are implicitly specified; an uncoupled train unit corresponds to an arrival at a depot, while a coupled train unit corresponds to a departure from a depot. Such arrival or departure events are associated with a specific train unit type. We assume that the depots of two different stations are independent of each other. In other words, the set of shunting movements at one depot is confined to that depot and has no impact on the shunting movements of another depot. We term a feasible solution to the TUSP a *shunting plan*, and this must satisfy two constraints. First, the total length (or capacity) of each individual depot track must not be violated at any given time; and second, no train unit ordering *conflicts* occur. A conflict occurs when the arrival of a train unit at a depot track blocks the departure of a train unit from the same track. In this paper, we assume that depot tracks function as last-in first-out (LIFO) stacks. This means that the last train unit to arrive at a depot track must be the first to leave. In reality, open ended depot tracks exist; however, such cases will also have ordering restrictions that must be obeyed. In this paper, any open-ended track is operated as a LIFO stack.

The TUSP is essentially comprised of two interdependent subproblems. A *matching* problem pairs the arrivals and departures of train units at a depot. This is necessary as only train unit types are known in the solution to the rolling stock problem. The resulting matching implicitly specifies how long individual units will spend in the depot and a track allocation problem or *parking* must be solved to identify on which track the units will be parked. Contrary to previous work in this field, we view the TUSP as a feasibility problem as the primary goal is to determine whether or not a given rolling stock schedule is feasible from a shunting perspective. Almost all of the operational cost is incurred in the rolling schedule and this schedule is unlikely to change in order to improve the combined objective of the TUSPs. It is important to simply know whether the generated shunting movements are feasible with respect to the depots. We note that after confirming feasibility of the shunting movements, the TUSPs could be resolved with an appropriate objective function. In the absence of a feasible solution, proving that no solution exists is equally important. If no solution exists, it means that the given rolling stock schedule is not feasible from a shunting perspective. As a consequence, in such situations, a new rolling stock schedule must be found. The emphasis in this paper is therefore on determining whether or not a solution to an instance of the TUSP exists.

In this paper we propose three new methods for solving the TUSP and benchmark their performance against several methods from the literature on both realistic and artificial instances. In particular we present a Constraint Program (CP) approach, a column generation procedure, and a greedy randomized heuristic approach. To our knowledge, CP has never been applied to the TUSP despite the fact that the methodology is extremely effective at solving feasibility problems. These are compared against a reference Mixed Integer Program (MIP) and a two stage approach from the literature. For the reference MIP we also experiment with a delayed constraint variant. The devised methods cover a broad range of optimization techniques. Some of the proposed approaches are exact solution methods, which find one feasible solution (if it exists) or prove that no solution exists. Others are heuristic approaches that strive to quickly find any feasible solution by searching subsets of the entire solution space and consequently cannot prove infeasibility. A comparison of all of the approaches on instances obtained from our industrial partners in The Netherlands and Denmark allows definitive conclusions regarding the strength and weaknesses of each approach, not to mention their applicability in practice, to be made. This paper therefore makes the following important contributions to the literature on train unit shunting

1. Three new methodologies for solving the TUSP
2. Extensive computational experiments that compare the performance of these new methods with existing methods from the literature
3. A benchmarking of the methods on problem instances from multiple railway operators in different countries.

In addition to the main contributions, this paper also makes several minor ones. These include: the development of several efficient initial infeasibility checks that can be run prior to solving an instance of the TUSP; a problem decomposition approach that reduces an instance of the TUSP to a number of smaller, independent TUSP subproblems; an extension of a reference MIP, giving it the potential to dynamically add constraints; and an extension of the TUSP to also allow parking at platform tracks within the station at the end of the planning horizon. Computational tests suggest that there is no single method that dominates all others.

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 time feasibility checks are discussed in Section 4 before introducing the new solution methods in Section 5. The benchmark solution methods are explained in Section 6. A special decomposition technique for the TUSP is shown in Section 7. The problem instances are presented in Section 8, followed by the benchmark of the solution methods. Finally, we conclude and give some remarks on further research in Section 9.

2 Literature Overview

To the best of our knowledge, the TUSP was first introduced in passenger railways by Freling et al. [2]. 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 matching of train units between arrivals and departures is given as input and not part of the problem. Otherwise, part of the problem is also to specify which compatible (arriving) train unit is matched to every departure. The remaining part of the TUSP is to find a valid parking plan for the in and out movements specified by the train matching. With the exception of Kroon et al. [8], 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. [2] consider the problem of parking train units overnight at a depot in such a way that each train unit can be retrieved, without moving others, 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 from the yard the next day. The problem is decomposed into two smaller subproblems: a matching problem and a parking (or track allocation) problem. The first entails matching arrivals and departures of train units at the depot under consideration. A solution to this problem hence also stipulates the train services each physical train 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 8.

The work of Lentink et al. [10] extends the work of Freling et al. [2], 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 depot to the station platforms. Their model is tested on small and big problems instances. 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. [5]. 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 train units arrive and 55 train units depart. The depot has 19 tracks and a total capacity of 4000 meters. Solutions to the problems are found quickly and the results are promising; however, only a single instance is considered.

Solving the TUSP without some form of matching/parking separation has been proposed by Kroon et al. [8]. The authors essentially extend the work of Freling et al. [2] and propose a large MIP formulation that simultaneously solves the matching and parking problem. The authors try to keep the depot tracks as homogeneous (with respect to the train 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 [6] present three different heuristics to solve a variant of the TUSP that includes maintenance scheduling. The model is tested on small instances and the run-times are low. Internal rearrangements are permitted if one train unit is blocking another train unit. The model is not tested on problem instances from practice.

In contrast to Freling et al. [2], Lentink et al. [10], and Haijema et al. [5], we propose several new methods that integrate the matching and parking problems. Our methods are compared with the integrated and sequential methods from the literature. Furthermore, unlike any other research in this area, we test all methods on several classes of large and realistic instances from different train operating companies. In addition, we present and discuss the possibility of parking train units at platforms at the end of the planning horizon. This can also be seen in Table 1, where an overview of existing literature is given.

Paper	Integrated	Method	Instances	Platform
Freling et al. [2]	No	MIP, CG	Nld	No
Lentink et al. [10]	No	MIP, CG	Nld	No
Haijema et al. [5]	No	DP	Nld	No
Kroon et al. [8]	Yes	MIP	Nld	No
Jacobsen and Pisinger [6]	Yes	H	Den	No
This paper	Yes	CP, MIP, H, (CG)	Nld, Den	Yes

Table 1: Overview of the contribution of papers on the TUSP. The first column denotes the paper. The second column indicates whether the matching and parking problems are integrated. The third column states which solution methods are used (CG stands for column generation, DP for dynamic programming, CP for Constraint Programming, MIP for Mixed Integer Programming, and H for heuristic). The fourth column states from which countries the practical settings come from. Here, Nld stands for Netherlands and Den stands for Denmark. The final column specifies whether or not overnight platform parking is taken into account.

3 Problem Description

This section begins with a formal description of the TUSP, and this is followed by an example problem instance. We then introduce the platform extension used in this paper. General notation is introduced throughout the section, and investigation into the complexity of the problem is also provided.

3.1 Problem description

The aim of the TUSP is to create a feasible shunting plan or prove that no such plan exists. A feasible shunting plan includes a feasible matching of all arrivals and departures of train units at the depot as well as a feasible assignment of the obtained matchings to tracks. A valid matching pairs all initially parked train units at the depot, as well as all train units that arrive during the day, with compatible departures. Any unmatched train unit must remain parked at the depot. In other words, all unmatched train units reside in inventory until the end of the planning horizon. Furthermore, in a feasible matching all departures must be covered by a train unit, otherwise the TUSP is infeasible. A feasible solution to the matching problem must be assigned to depot tracks in a conflict-free manner. Without loss of generality, we can assume there is a one-to-one correspondence between arrivals and departures. Initially parked trains can be modelled as early arrivals, and train units that remain at the depot at the end of the planning horizon can be modelled as late departures.

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

A solution to the parking problem 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 train unit on a track must be the first train unit to leave. A train unit cannot leave a track (for a departure) if a different train unit has arrived in the meantime and is staying on the same track. Otherwise, a conflict would occur. We assume that an assigned track is occupied 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. By using this conservative approach, we are guaranteed to find a solution that is feasible in practice.

We make several assumptions in this paper. First, the shunting movements at different stations in the railway network are assumed to be completely independent of each other.

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
Departure	a	t_∞
Departure	b	t_∞

Table 2: Example list of events in a problem instance. Note, that a_1 and a_2 (and b_1 and b_2) denote the same train unit type.

Hence, the respective TUSPs can be solved separately for every station. Second, no maintenance operations are considered in our problem. Thus, train units of the same type are assumed to be completely interchangeable. Third, internal shunting of train units is not considered; once a train unit is parked it can only be moved once 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 on a depot 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 the lowest value, $\beta = 1$.

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

In contrast to the literature, no cost structure is defined in our work on the TUSP. The TUSP is considered a feasibility problem as this is the most important question to answer. Any of the proposed solution approaches could 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 quickly detect feasibility. We note, however, that almost all of the presented approaches can, without much difficulty, be extended to include an objective function.

3.2 Example

An example of a problem instance is given in Table 2. The example highlights the importance of integrating the matching and parking problems. The table lists a set of arrival and departure events. The time each event occurs and its associated train unit type are also specified.

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

Table 3: Two feasible solutions to the matching problem for the TUSP instance in Table 2.

Furthermore, for this example, the lengths of the train unit types for a, b and c are 200, 100 and 150 meters, respectively. Two depot tracks are assumed to be available, one with length 550 meters and one with length 200 meters.

Each arrival must be matched to a compatible departure, or remain in the depot. For the latter option the arrival is paired to an artificial event, which is denoted t_∞. For the sake of argument, if parking constraints are ignored, we can identify two feasible solutions to the matching problem. These are given in Table 3.

Furthermore, without taking matching constraints into account there are in total three feasible parking possibilities. Figure 1 denotes the first parking possibility. The second parking possibility can be achieved by switching train units a_1 and a_2. Finally, Figure 2 shows the third parking possibility. Matching 1 is infeasible with respect to all three parking possibilities. In the parking given in Figure 1 the departure of train unit b_1 is blocked by train units c and b_2. In the parking given in Figure 2 the departure of train unit b_1 is blocked by train unit b_2. Matching 2, on the other hand, is conflict-free in all shown parking possibilities. This is because all train units can depart at their specified time without being blocked by other train units. Thus, only matching 2 has at least one feasibile parking possibility. If matching 1 would be the resulting matching after solving the matching problem independently, then the parking problem simply be infeasible.

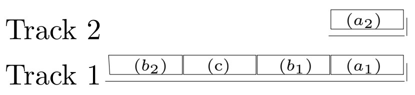

Figure 1: Feasible parking 1

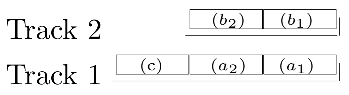

Figure 2: Feasible parking 2

Platform Parking

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

According to some railway operators, e.g. Netherlands Railways (NS) and Danish State Railways (DSB), train units are, under certain circumstances, parked on platform tracks. The rolling stock activities for the following day are known, and it can be practical to park train units on a platform track overnight if the parked composition is to depart early the next day. Depending on the track layout and the number of platforms at a station, a number of platform tracks may be eligible to be used in this way. However, still, a number of platform tracks must 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 train services (of the following day) dictate which train units can be parked on the N available platform tracks. A train service may consist of multiple train units, allowing more than a single train unit to be parked on a platform track. For instance, if $N = 2$, and the first two train services have the compositions aa and bc, then two train units of type a can be assigned to the first platform track and a single train unit of each type b and c to the second platform track. In this 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, a matched arrival and departure pair can only be assigned to a 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. Note that in this paper we only assume that the train units present in a composition of the first train services of the next day can be parked on a platform, but we do not restrict them to be parked in the correct composition order. For instance, the composition of the first train service bc can be parked in the order bc, but also in the order cb. We assume that the correct order can be fixed during the night.

3.3 General Notation

When solving an instance of the TUSP for a specific depot, a set of arrival and departure events is assumed to be given. This set of events is denoted by E. The sets E^{arr} and E^{dep} respectively contain all arrivals and departures, and together define a partition of all events; that is, $E^{arr} \cup E^{dep} = E$ and $E^{arr} \cap E^{dep} = \emptyset$. An arrival corresponds to a train unit that is uncoupled at the station and must be parked in the depot, while a departure corresponds to a train unit that is to be coupled at the station and must be retrieved from the depot. Furthermore, we are given a set, M, of train unit types and a set, S, of tracks on which units can be parked. Two subsets of S, S_d and S_p, with $S_d \cup S_p = S$ and $S_d \cap S_p = \emptyset$ contain, respectively, the set of depot tracks and the set of platform tracks. All definitions are summarized in Table 4. The time at which event $e \in E$ takes place is denoted by t_e, and $m_e \in M$ denotes the train unit type of the corresponding event. We let c_s denote the 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 denoted by l_m. Table 5 summarizes these parameters.

Complexity

The complexity of the TUSP has been addressed by multiple authors in the literature. Several variations of the problem exist, each of which is known to be $\mathcal{NP}$-hard. The shunting problem considered by Freling et al. [2] is essentially a specialization of the considered TUSP. They prove the variant to be $\mathcal{NP}$-hard by a reduction from the Tram Dispatching Problem studied by Winter [14], 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 a reduction from the Graph Coloring Problem (GCP). In the GCP a color must be assigned to each vertex of a graph such that no adjacent vertices have the same color. Two

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 4: List of defined sets

Notation	Description
t_e	Time event $e \in E$ takes place
m_e	Train unit type corresponding to event $e \in E$
l_m	Length of train unit type $m \in M$
c_s	Capacity of track $s \in S$

Table 5: List of parameters

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 Garey et al. [3].

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 constraints 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 the resulting parking is conflict-free. Second, we argue that the TUSP is a generalization of the GCP. Given an 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 train unit type to all train units. A vertex (in the GCP) corresponds to a track assignment of a matching, and the selected track represents the selected color. The time of the constructed events reflects an ordering that corresponds to the edges in the GCP, i.e., if two vertices are connected then the corresponding arrival time is in between the arrival and departure time of the other. Furthermore, the departure time is later than the departure time of the other. Consequently, the two train units are conflicting. If the constructed TUSP instance contains a feasible solution, then a feasible assignment of k colors is given by the track assignments of the matchings of the TUSP instance. Thus, the TUSP can be reduced to an instance of the GCP. An additional necessary condition is that the resulting graph is not an interval graph, because there exist polynomial algorithms to solve the interval coloring problem. Our graph is not an interval graph as two train units are not necessarily in conflict if they have overlapping intervals of time in the depot. Consider Figure 3 as an example, this depicts five train units. The interval of time each train unit spends in the depot is indicated by a horizontal line, starting at the unit's arrival time at the depot and finishing at the unit's departure time from the depot. Figure 4 shows the resulting conflict graph. Here t1 represents train unit 1, t2 train unit 2, etc. Note that train units three and four are not in conflict. This would not be the case in an interval graph. Thus we can conclude that the TUSP is $\mathcal{NP}$-hard.

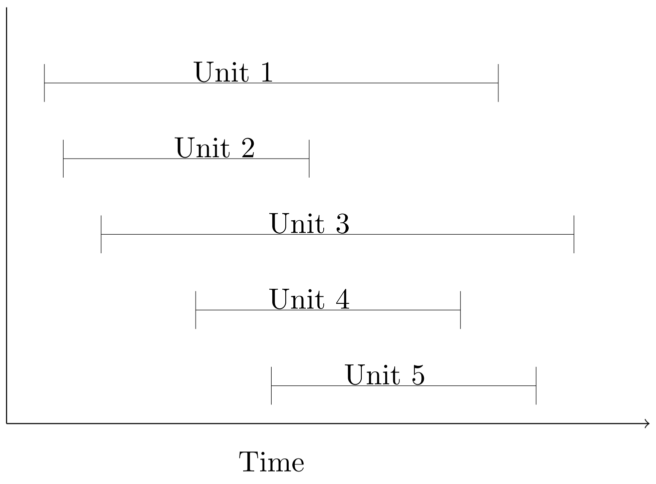

Figure 3: Interval graph

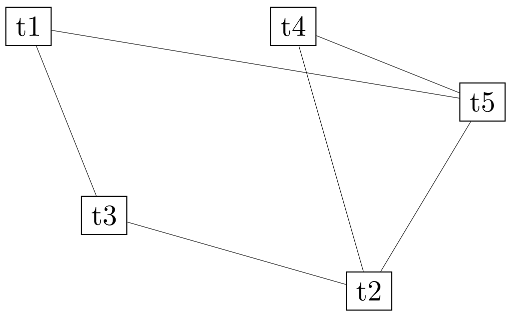

Figure 4: Conflict graph

4 Infeasibility Checks

A select number of efficient checks can be performed independently of any solution method to assess the feasibility of an instance of the TUSP. The problem instances considered in the benchmark testing of Section 8 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 at least the sum of the lengths 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 with respect to the capacity constraints of all tracks. At the start of the planning horizon a feasible parking must exist, otherwise no solution can exist to the TUSP. With a given set of initial train 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 train unit. Each depot track corresponds to a knapsack, where the capacity is equal to the track length. Good algorithms for solving the MKP exist (see e.g. Pisinger [13]).

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

Feasible Matching

In the common case, multiple feasible matchings exist 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 [12]), which is solvable in polynomial time as the resulting matrix of the Linear Program (LP) is totally unimodular. As the number of arrival and departure events are the same, the problem is the linear AP; this can be solved using specialized polynomial time algorithms, such as the Hungarian algorithm (Kuhn [9]). Again, as our problem instances are the result of a feasible rolling stock schedules at least one matching must exist.

5 Solution Methods

In this section we describe three new solution methods for solving the TUSP. Section 5.1 is dedicated to the Constraint Programming Method (CPM), while Section 5.2 introduces the column generation method. An overview of the Randomized Greedy Construction Heuristic (RGCH) approach is given in Section 5.3. Existing methods in the literature are described in Section 6.

5.1 The Constraint Programming Method

As the TUSP is essentially a feasibility problem, a CP approach might be effective. Our proposed CP formulation is inspired by the rolling stock composition model of Fioole et al. [1]. This formulation was originally used for Rolling Stock (re-)Scheduling; however, we can use the idea of compositions and composition changes for the TUSP. In this formulation we attempt to assign compositions to tracks whenever an event occurs. At such times one composition must be assigned to each track individually. Similar to the rolling stock problem, a composition consists of a number of train units in a specific order. For instance, in Figure 2, the composition *caa* is assigned to track one and the composition *bb* is assigned to track two. Note that the empty composition, containing no train units, is a valid composition as well. Composition changes can only occur whenever events occur. A composition change is the transition of one composition to another. For instance, if an arrival of a train unit takes place at a specific track, then a train unit is, in effect, coupled to the composition currently assigned to the track. Thus, a composition change consists of two compositions, the one prior to the event occurring and the one immediately after it has taken place.

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 train unit of type a is arriving, then only composition changes where a train unit of type a appears on top of

Set or parameter	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 $s \in S$
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 train 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 train unit of type $m \in M$ is removed from the composition on the track during composition change $q \in Q$

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

the stack are included in $Q_{e,s}$. Additionally, composition changes where no train units are appended or removed are included as all unaffected tracks remain unchanged.

The CP also requires some parameters. First of all, i_s 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 notation $\texttt{In}[q]$ and $\texttt{Out}[q]$ to denote the index of the first and second composition in a composition change in the list of allowed composition changes, i.e. the original composition and its successor composition. Finally, $\alpha_m[q]$ and $\beta_m[q]$ specify whether a train unit of type m is appended or removed. Table 6 summarizes the sets and parameters used in the CP approach.

We define two families of decision variables. First, the integer variable $X_{e,s}$ specifies which composition is assigned to 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 has occurred $e \in E$ 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 to track s after event e occurs, then it means that train unit d was parked there first, followed by train units c, b, and a.

Similarly, the integer decision variable $Y_{e,s}$ indicates the composition change that takes place on track $s \in S$ just after the occurrence of event $e \in E$. An example of this is the composition change from aa to a when one train unit of type a departs track s. Again, an integer value corresponds to a transition from one composition to another.

The construction of $Q_{e,s}$ models the allowed composition changes with respect to the depot track capacity and the LIFO restrictions. Note that platform parking can also be modeled in the 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 train unit type of events are considered. First, if event $e \in E$ is an arrival of a train unit of type a, then only composition changes where a train unit of type a is appended and composition changes where no train 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 occurs, then there are only two allowed composition changes after a departure event $e \in E$ on track s: $abcd \to abcd$, or $abcd \to bcd$. train units b, c, and d cannot depart as train unit a is blocking them. Finally, with respect to the platform tracks, it is not allowed to park train units at platform tracks during the day, the only allowed composition change after events during the day is *empty* $\to$ *empty*. For the last events of the day, however, platform track 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 the following day. This can be considered by only allowing composition changes that involve transitions to compositions which are a subset of the departing train service composition.

The TUSP can therefore be modeled with the following mathematical program, which is used as a basis for the CPM:

$$X_{\lambda_e,s} = \mathtt{In}[Y_{e,s}] \qquad \forall e \in E, s \in S \tag{1}$$

$$X_{e,s} = \mathtt{Out}[Y_{e,s}] \qquad \forall e \in E, s \in S \tag{2}$$

$$\sum_{s \in S} \beta_m[Y_{e,s}] = 1 \qquad \forall e \in E^{dep}, m \in M : m_e = m \tag{3}$$

$$\sum_{s \in S} \alpha_m[Y_{e,s}] = 1 \qquad \forall e \in E^{arr}, m \in M : m_e = m \tag{4}$$

Constraints (1) 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 has occurred. 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 the track after the composition change took place, which equals the composition that is assigned to the track just after the occurrence of event e. This is modeled by Constraints (2).

Every departure event has a corresponding train 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 train unit of type m_e has departed; on all other tracks no shunting movements can take place. This is modeled by Constraints (3). A similar requirement holds for an arrival. The corresponding train unit m_e has to be appended to precisely one track; this is handled by Constraints (4). 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.

5.1.1 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 Fioole et al. [1], the model does not scale well. Long depot tracks and a high number of train 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. Many different train unit types further increase the combinatorial solution space.

Preliminary results showed that small instances with two train unit types are practical to solve. However, larger instances with four train unit types quickly become impractical to solve, primarily due to the memory footprint. As a remedy, the CPM can be solved in a more practical way.

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

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 train 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 $\epsilon = |M|$.

5.2 Column Generation

Column generation is a well-known decomposition method for solving large-scale LPs. The most attractive feature of the approach is that it only generates variables that have the potential to improve the objective function while implicitly considering all non-basic variables included in the formulation. In what follows, we present a column generation procedure for the TUSP. To apply column, we decompose the problem by track, and attempt to assign each track a set of possible matchings, termed a *matching pattern*. A matching pattern is a subset of matchings that can be feasibly assigned to 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 the dynamic generation of variables that represent promising matching patterns. In this paper, the solution method is referred to as the Column Generation (CGM).

The proposed formulation is based on the methodology presented by Freling et al. [2], 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. For 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 track parking and LIFO constraints are already satisfied in this set. 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 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 exactly 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. The objective of the model is to match as many arrivals and departures as possible. Thus, the objective function simply minimizes the number of unassigned arrivals. The binary parameter $\alpha_{e,p}$ is used to indicate 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.

$$\text{Minimize: } \sum_{e \in E^{arr}} Y_e \tag{5}$$

subject to:

$$\sum_{p \in \mathcal{P}_s} X_{p,s} \leq 1 \qquad \forall s \in S,\ (\boldsymbol{\mu}) \tag{6}$$

$$\sum_{s \in S} \sum_{p \in \mathcal{P}_s} \alpha_{e,p} X_{p,s} + Y_e = 1 \qquad \forall e \in E^{arr},\ (\boldsymbol{\pi}) \tag{7}$$

$$\sum_{s \in S} \sum_{a \in \mathcal{P}_s} \alpha_{e,p} X_{p,s} + Z_e = 1 \qquad \forall e \in E^{dep},\ (\boldsymbol{\gamma}) \tag{8}$$

$$X_{p,s} \in \{0,1\} \qquad \forall s \in S, p \in \mathcal{P}_s, \tag{9}$$

$$Y_e \in \{0,1\} \qquad \forall e \in E^{arr}, \tag{10}$$

$$Z_e \in \{0,1\} \qquad \forall e \in E^{dep}. \tag{11}$$

The objective function (5) minimizes the number of uncovered arrivals. Constraints (6) ensure each depot track is assigned at most one matching pattern. Constraint sets (7) and (8) 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 (9)-(11). We refer to Model (5)-(11) 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 is included. We relax the integrality restrictions and associated bounds given by (9)-(11). 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, or subproblem, is solved to determine if any favourable matching patterns exist. Promising variables are inserted iteratively into the restricted master problem until none exists - 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 (5)-(11) 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 Lübbecke and Desrosiers [11].

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 graph.

The directed, acyclic graph contains one node for every possible matching, one node for every arrival (corresponding to not parking the arrival), and a source and sink node. It 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 - provided 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. The cost on an any arc entering a node is equal to the reduced cost contributions of the associated matching. For example, 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 5

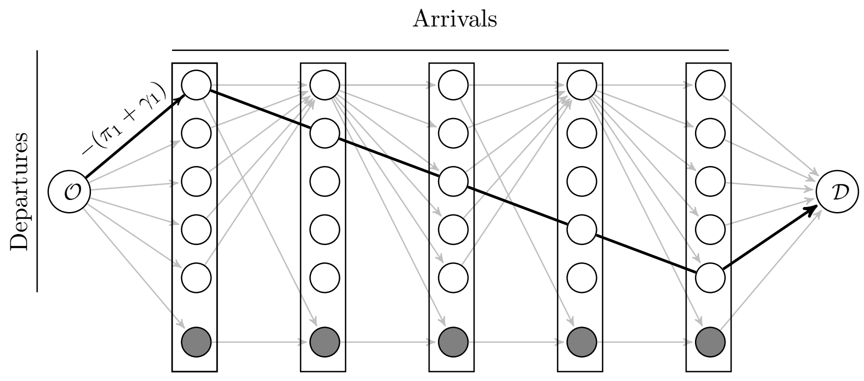

Figure 5: An example subproblem network with five arrivals and five departures. Every node corresponds to either the matching of an arrival and a departure or an unmatched arrival (gray nodes). All paths originating at the source ($\mathcal{O}$) and terminating at the sink ($\mathcal{D}$) represent a 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.

As we must observe the available track length and satisfy the LIFO requirements 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. [2]; however, as we must also simultaneously find the matchings, the proposed network is much bigger. Additionally, we must also keep track of previously matched departures. This is not nice from a dominance perspective in a label setting algorithm as it weakens the possibility of dominating partial paths away in the shortest path solve. This has a detrimental affect on the runtime of the shortest path algorithm. For large problems it is not even practical. We implement the dominance strategy described in the label setting algorithm of Freling et al. [2] and include an extra check in the extension of a partial path to make sure that we do not visit a matching for a previously matched departure. This is a heuristic variable generation procedure, but keeps the runtimes 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 6.1, 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.

5.3 The Randomized Greedy Construction Heuristic

Modeling ordering constraints efficiently in integer LPs such as LIFO constraints is cumbersome. The proposed solution methods and all existing ones overcome this issue by either adding all pairwise conflicts, enumerating all possible transition states, or by generating feasible parking patterns. As such, all methods have scaling issues, whether it be in the number of constraints or the number of variables. In contrast, modelling one or multiple stacks programmatically would not have such issues.

We propose a heuristic that greedily assigns arrivals and departures to and from tracks. The important aspects of this heuristic are the efficiency of the construction and the randomization of the greedy choice. Together these characteristics allow the heuristic method to try many different 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 $M$ and parking plan
4: $M \leftarrow \emptyset$
5: $S \leftarrow$ InitializeEmptyTrackStacks()
6: for $e \in E$ do
7:     if Type($e$) =Arrival then
8:         $s \leftarrow$ FindRandomCompatibleTrack($S$)
9:         if $s = \emptyset$ then
10:            return $\emptyset$
11:        else
12:            $S[s] \leftarrow$Push($S[s], e$)
13:    else
14:        $s \leftarrow$ FindRandomCompatibleUnitType($S$)
15:        if $s = \emptyset$ then
16:            return $\emptyset$
17:        else
18:            $(S[s], e') \leftarrow$Pop($S[s]$)
19:            $M \leftarrow M \cup \{(e, e', s)\}$
```

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 chronologically. When an arrival event occurs, 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 train unit has the same train unit type

2. A track which is empty
3. Any track with sufficient capacity

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

For a departure event, i.e. a train unit must leave the depot, tracks are processed in a random order. The track with the correct train unit type (on top of the stack) is selected. Here it might be worthwhile considering a selection criteria approach based on the effect of removing this train 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 event, its matching departure event, and a specific track to which the corresponding train unit is appointed. On arrival the train unit is parked on the track, and the specified departure extracts the train 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 a predefined time limit is reached.

6 Literature methods

In the previous section three novel methods were introduced to solve the TUSP. In Section 8, we compare these methods with existing methods from literature. For completeness, we briefly describe two methods based on existing approaches. Section 6.1 is dedicated to the Reference MIP Method (RMM), while Section 6.2 focuses on theTwo-Stage Method (TSM).

6.1 The Reference MIP Method

We begin by explaining the RMM. This model is based on the paper Kroon et al. [8]. Here, 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 train 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 event is denoted by $\mathcal{A}^{arr}_{e_1}$ and the set of matchings where event $e_2 \in E^{dep}$ is the departure event is denoted by $\mathcal{A}^{dep}_{e_2}$. These sets are summarized in Table 7.

We change the definition of X here, such that $X_{a,s}$ takes a value of 1 if and only if matching $a \in \mathcal{A}$ is selected and parked on track $s \in S_d$.

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 (12). Recall that a

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

Table 7: List of MIP specific sets

departure represents an actual departure or a stay in the depot. Similarly, each departure must be assigned to exactly one arrival, c.f. Constraints (13).

$$\sum_{s \in S_d} \sum_{a \in \mathcal{A}_e^{arr}} X_{a,s} = 1 \qquad \forall e \in E^{arr} \tag{12}$$

$$\sum_{s \in S_d} \sum_{a \in \mathcal{A}_f^{dep}} X_{a,s} = 1 \qquad \forall f \in E^{dep} \tag{13}$$

Constraints (12)-(13) ensure a feasible matching. For the parking, the capacity of a depot track can not be exceeded at any point in time. It is sufficient to ensure that the track capacity is not exceeded at every arrival, c.f. (14).

$$\begin{aligned} &\sum_{\{e' \in E^{arr} | t_{e'} \leq t_e\}} \sum_{a \in \mathcal{A}_{e'}^{arr}} X_{a,s} \cdot l_{m_{e'}} \\ - &\sum_{\{e' \in E^{dep} | t_{e'} \leq t_e\}} \sum_{a \in \mathcal{A}_{e'}^{dep}} X_{a,s} \cdot l_{m_{e'}} \leq c_s \qquad \forall e \in E^{arr}, s \in S_d \end{aligned} \tag{14}$$

For each arrival, Constraints (14) 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 forbid such assignments, c.f. Constraints (15). It states that any two pairs of matchings cannot be assigned simultaneously if they block each others' movements.

$$X_{a,s} + X_{a',s} \leq 1 \qquad \forall s \in S_d, (a, a') \in \mathcal{C} \tag{15}$$

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. [8]. In general, the problem of finding maximum cliques is $\mathcal{NP}$-hard(Karp [7]). 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 train units are modeled using arrivals and that parking (at the end of the planning horizon) is modeled using departures. Any initial train unit to track assignment can be modeled by fixing the variables corresponding to the first set of events. In the computational experiments of Section 8, an initial parking order is imposed. We note that in the implementation of the model, all fixed variables are removed to obtain a more compact model.

Extension

Due to the large number of conflict constraints (15) present in the model, we introduce in an extension of the reference MIP, termed the Delayed Constraint MIP Method (DCMM), in which Constraints (15) are generated on-the-fly. The DCMM is solved as a MIP model, where violated conflict constraints 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.

6.2 The Two-Stage Method

In the Two-Stage Method (TSM) the matching and parking problems are solved sequentially. Given a feasible matching to an instance of the TUSP, the remaining problem simply entails assigning the matchings to tracks. 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 and motivates the TSM, where in the first stage a feasible matching of arriving and departing train units is generated, while in the second stage the method tries to assign the found matchings to tracks. The TSM is based on the method of Freling et al. [2]. One important difference between our approach and that described in Freling et al. [2] is that we consider individual units in isolation; when parking units that arrive on the same train no attempt is made to park them on the same track. Similarly, if two units are needed for a departing train service, they can be retrieved from different tracks.

The matching and the parking problems can be solved using different methods. For the matching problem we use 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. [2] propose 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 we reformulate the binary decision variable X_a; this variable 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.

$$\sum_{a \in \mathcal{A}_e^{arr}} X_a = 1 \qquad \forall e \in E^{arr}, \tag{16}$$

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

$$X_a \in \{0, 1\} \qquad \forall a \in \mathcal{A}. \tag{18}$$

There is no objective used in the MIP, since we are interested in feasibility only. Constraints (16) and (17) ensure, respectively, that each arrival and departure event appears in exactly one matching. The variable domain is given by (18). We note that several solutions (i.e. matchings) to the model may exist. It is therefore possible to guide the solution in a more advanced approach, but we do not pursue this addition in this paper.

If a solution to the matching problem problem exists, we proceed to the second stage and attempt to assign them to depot tracks. For this, we use the MIP described by Haahr et al. [4], which is identical in structure to the reference MIP approach described earlier.

The TSM is similar to what is described by Freling et al. [2]. However, the matching problem we propose is slightly different. In our formulation we define a decision variable for each feasible (arrival, departure) pair, while in Freling et al. [2] the decision variables focus on individual units and ensure feasible matchings are obtained through constraints. Furthermore, the model of Freling et al. [2] attempts to keep train units together as much as possible. We are only interested in feasibility. Finally, Freling et al. [2] propose a heuristic column generation procedure to solve the resulting parking problem. We instead use a standard MIP solver.

7 Type and Track Decomposition

Some problem instances contain many events, many train 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, because they contain an explicit representation of the problem.

The solution space can be reduced significantly by decomposing the problem instances by train unit types and tracks. In the proposed decomposition, a train unit type is restricted to park on a select subset of tracks. The partitioning of the tracks and train 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 train unit types and tracks are partitioned into K groups, such that one group of train unit types is assigned to one group of tracks. By construction, no interaction needs to take 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 explained and tested in Section 8. 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. In further research it could be interesting to investigate how to select good partitions.

8 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 Netherlands Railways (NS) - the principal operators in Denmark and the Netherlands respectively.

All instances, except the DSB and ARTIFICIAL 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. For the DSB class all arriving and departing events are explicitly given by the operator.

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. In total there are thus 110 instances.

The NS-HARD class is artificially constructed from the NS class, where fewer tracks are available at busy stations. These artificial cases are therefore more constrained in terms of capacity, in turn reducing the number of feasible parking plans. These problem instances have been included in an attempt to stress test the solution methods.

All computational 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.

CPM A CP formulation inspired by the composition model in Fioole et al. [1]. The formulation is solved using the CPLEX constraint program solver.

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 the instances: The first column indicates to which class the problem belongs. The second column defines the station. The third and fourth columns 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 train unit 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 the 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 the instances.

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

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

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

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*.

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 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 LP relaxation was significantly greater than the time the MIP based approaches took 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 Haahr et al. [4], where a similar column generation approach was outperformed by a MIP approach for the parking problem only. We have retained a discussion of the approach in this paper as it provides a more complete overview of the methodologies tested. We believe the negative results obtained with this procedure are still valuable when comparisons are made with the other approaches.

Tables 11 and 12 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 12 shows the average runtimes per problem class per method.

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 a feasible solution for those 94 instances. The TSM was unable to find a feasible solution for one of the instances. The average solution time is less than one second for all methods.

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

The CPMH, RGCH and TSM perform well on all 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 on average. However, for the NS-HARD 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).

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

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

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

Table 12: Average runtimes for finding solutions grouped by method

Table 13 shows the number of instances for which infeasibility is proven per method within the time limit. There were two infeasible instances in total. In both cases infeasibility was detected by all exact approaches and the CPMH. The heuristic method RGCH is by definition not able to prove infeasibility. The TSM can only prove infeasibility if there is no feasible matching. However, there is at least one feasible matching for both instances. Consequently, TSM was unable to prove infeasibility here.

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 train unit types. Recall that the TSM and the RMM are only based on proposed methods in the literature. While not identical, they are very similar in structure and do suffer from the same limitations. The focus of this paper is not on improving upon these existing methods, but rather on comparing the strengths and weaknesses of different modelling and/or solution approaches for the TUSP. With any two stage approach for the matching and parking, due to its lack of integration, feasibility in the second stage cannot be guaranteed. We will show this for the proposed TSM in Section 8. Furthermore, a MIP based approach is unlikely to scale well. We have suggested a delayed constraint generation approach to remedy this here, and this method is definitely an improvement. Instead of

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

Table 13: Number of instances proved to be infeasible by the methods.

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

Table 14: Summary of results achieved when running different methods on problem instances with overnight parking. The columns respectively show the instance considered, the number of problem instances, and the number of solved instances for every method.

dynamically generating violated constraints, clique constraints can be used to strengthen the formulation (as is the case in Kroon et al. [8]). However, as we have already mentioned, identifying and/or separating cliques is not an easy problem. The results suggest, rather emphatically, that MIP based appraoches struggle to solve large instances of the TUSP.

In the following two sections we show results of using two different extensions. First, in Section 8 we test the performance of all proposed models when overnight parking is allowed. Secondly, in Section 8 we test the decomposition approach as explained in Section 7.

Overnight Parking

In the common case all train units leave the depots during the early hours and enter the depot at the end of the day. Some train units enter and leave the depots during the day, e.g. before and after the rush hour periods. The capacity of the 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 train 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 at platforms tracks.

Tables 14 and 15 show the results of the overnight parking instances on four different stations. 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 when using overnight parking differ from the results with the common instances. The TSM now performs worse than the other methods. Remember that the TSM first finds one feasible matching and tries to assign the resulting matchings to tracks. Fixing the matching in an 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.

Furthermore, the results show that all other considered methods, except the DCMM, can efficiently solve all instances. Remember that all these methods integrate the matching and parking problem. As such, these models do not restrict flexibility when finding a solution.

The average runtimes shown in Table 15 reveal that the CPMH is faster than the DCMM in general when considering these extended instances. We note, however that these instances

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

Table 15: Summary of the average runtimes achieved when running different methods on problem instances with overnight parking. The columns respectively show the instance considered, the number of problem instances, and the average runtimes.

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

Table 16: Summary of results achieved when running different methods on problem instances decomposed by splitting tracks and train unit types. The columns respectively show the instance considered, the number of decompositions, the number of feasible decompositions, the number of generated subproblems, and finally the number of **F**easible, **I**nfeasible and **T**imed-out instances for every method.

only contain 2-4 different train unit types. The performance of the CPMH is expected to decrease with higher numbers of different train unit types.

Track Splitting

As explained in Section 7, the TUSP can be decomposed into several independent problems by partitioning the full problem by train unit types and tracks. 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, where this number is smaller than the number of train unit types and/ or tracks available. Second, tracks and train unit types are assigned randomly to the available groups. If any group has an empty set of train unit types or tracks, then the whole generation is rejected. Further, it is ensured that the maximum depot capacity required by the train unit types is less than the capacity of the tracks in the group. Finally, if train units are positioned initially in the depot, then this naturally adds constraints to the generation of the groups (e.g. if train units of different types are on the same track initially in the depot, then those two train unit types and the track where they are parked on are in the same group).

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

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

Table 17: Summary of average runtimes achieved when running different methods on problem instances decomposed by splitting tracks and train unit types. The columns respectively show the instance considered, the number of decompositions and finally the average runtime for all found solutions.

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 in the CP model since the resulting number of train unit types is decreased. We do not present the result of the decomposition method on all instances, but only on these selected instances. First of all, applying the decomposition method on instances with few tracks and/or train unit types is not beneficial. Secondly, the same conclusions can be drawn from the other large NS instances as from the three NS instances discussed in this section. Finally, the same conclusions can also be drawn from the resulting five DSB instances as from the two DSB instances we discuss here.

An overview of the generated instances and results is shown in Table 16. The average runtimes are listed in Table 17. The HFDO, GVC and RTD instances were successfully decomposed, as all the resulting subproblems were feasible. The considered solution methods were able to 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 efficiently, except for one. The CPMH is 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 not to be efficient 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 computation time is observed for both the DCMM and the RGCH in Table 17.

Using a decomposition of track and train unit types is shown to be worthwhile investigating. It is out of the scope of this paper to look further into different decomposition techniques. We do acknowledge that this could be an interesting future research direction.

8.1 Artificial instances

A fifth class ARTIFICIAL, summarized in Table 18, represents instances which are artificially generated. The data for these instances has been made publicly avaialble[1]. By doing this we hope to encourage other researchers to develop models for the TUSP as well.

[1]Available online at `http://www.ms.man.dtu.dk/research/instances`

Class	Events	Tracks	Length	Types
ARTIFICIAL1	3692	9	5 800	4
ARTIFICIAL2	537	9	5 800	4
ARTIFICIAL3	509	9	5 800	4
ARTIFICIAL4	2096	9	5 800	4
ARTIFICIAL5	2386	9	5 800	11
ARTIFICIAL6	326	9	5 800	11
ARTIFICIAL7	594	9	5 800	11
ARTIFICIAL8	892	9	5 800	11

Table 18: Summary of artificial instances.

Instance	CP	CPMH	RGCH	DCMM
ARTIFICIAL1	-	-	1.3	-
ARTIFICIAL2	38.3	32.3	0.1	-
ARTIFICIAL3	45.2	3.2	0.1	-
ARTIFICIAL4	-	24.2	0.3	-
ARTIFICIAL5	-	-	-	-
ARTIFICIAL6	-	6.261	0.1	133.4
ARTIFICIAL7	-	26.3	0.1	-
ARTIFICIAL8	-	16.9	0.2	-

Table 19: Summary of runtimes achieved on the Artificial instances. The columns respectively show the instance considered, the runtime for the found solutions by the methods. ”-” denotes that the method timed out without finding a feasible solution or proving infeasibility.

Table 19 shows the results of testing the CP, CPMH, RGCH, and DCMM on the artificial instances. As can be seen, both heuristic methods perform best; however, instance ARTIFICIAL5 cannot be solved by any of the methods within time.

9 Conclusion

In this paper we develop and benchmark 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 feasible shunting plan. Three novel solution approaches are proposed: a CP formulation, a column generation approach, and a greedy construction heuristic. These new methods were compared with two methods from the literature. 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 track 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, demonstrated that the CGM was outperformed by all other methods. Furthermore, it revealed the shortcomings of exact models (RMM and CPM). 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 train unit types. In turn, the DCMM can solve the instances with a high number of train 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.

The TUSP can be decomposed by splitting the available tracks and train unit types into several independent and smaller subproblems. A subset of the larger instances were decomposed in a second benchmark. In general, most of 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. The results show that the integrated methods are still able to solve these problems in reasonable time. However, the TSM was 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 be short sighted, leading to 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, then the DCMM and the CPMH can be adopted in a subsequent phase.

This paper focuses only on finding a feasible solution. No distinction is 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 future work on the TUSP. Other practical aspects need to be taken into account as well, e.g. detailed routing of the train units through the depot, parking whole compositions instead of single train units, train 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] Pieter-Jan Fioole, Leo Kroon, Gabor Maroti, and Alexander Schrijver. A rolling stock circulation model for combining and splitting of passenger trains. *European Journal of Operational Research*, 174(2):1281–1297, 2006.

[2] Richard Freling, Ramon M Lentink, Leo G Kroon, and Dennis Huisman. Shunting of passenger train units in a railway station. *Transportation Science*, 39(2):261–272, 2005.

[3] Michael R Garey, David S Johnson, and Larry Stockmeyer. Some simplified np-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63. ACM, 1974.

[4] Jørgen Thorlund Haahr, Richard Martin Lusby, Jesper Larsen, and David Pisinger. *Simultaneously recovering rolling stock schedules and depot plans under disruption.* Proceedings of the 13th Conference on Advanced Systems in Public Transport (CASPT), 2015.

[5] René Haijema, CW Duin, and Nico M. Van Dijk. Train shunting: A practical heuristic inspired by dynamic programming. *Planning in Intelligent Systems: Aspects, Motivations, and Methods*, pages 437–475, 2006.

[6] Per Munk Jacobsen and David Pisinger. Train shunting at a workshop area. *Flexible services and manufacturing journal*, 23(2):156–180, 2011.

[7] Richard M Karp. *Reducibility among combinatorial problems.* Springer, 1972.

[8] Leo G Kroon, Ramon M Lentink, and Alexander Schrijver. Shunting of passenger train units: an integrated approach. *Transportation Science*, 42(4):436–449, 2008.

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

[10] Ramon M Lentink, Pieter-Jan Fioole, Leo G Kroon, and Cor van't Woudt. Applying operations research techniques to planning of train shunting. *Planning in Intelligent Systems: Aspects, Motivations, and Methods*, pages 415–436, 2006.

[11] Marco Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations Research*, 53:1007–1023, 2004.

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

[13] David Pisinger. *Algorithms for Knapsack Problems.* PhD thesis, DIKU, University of Copenhagen, Denmark, 1995. Technical Report 95-1.

[14] Thomas Winter. *Online and Real-Time Dispatching Problems.* PhD thesis, Technical University, Braunschweig, Germany, 1999.