CASPT 2015

Simultaneously Recovering Rolling Stock Schedules and Depot Plans Under Disruption

Jørgen Thorlund Haahr · Richard Martin Lusby · Jesper Larsen · David Pisinger

Keywords Depot Planning · Disruption Management · Integration

Abstract In this paper we consider two important railway optimization problems. In particular, we focus on the Rolling Stock Rescheduling problem and the Depot Replanning problem, respectively. We present an integrated framework for solving these two problems simultaneously, and show that it is fast enough to be applied in a disruption recovery setting. Furthermore, we provide a comparison of several solution strategies to the Train Unit Parking Problem, and, by way of an example prove the heuristic nature of a previously proposed optimal approach. We analyse the performance of the proposed methodology on a number of artificial data sets as well as several real-life case studies provided by DSB Stog, a suburban train operator in the greater Copenhagen area.

1 Introduction

Disruptions are an undesirable yet inevitable part of the day-to-day planning for a railway company. Often they will render the planned, optimized schedules infeasible and, if handled poorly, can propagate through the network with a host of unwelcome consequences, both for the train operating company itself and its passengers alike. Therefore, ensuring that the impact of the disruption is as small as possible is of utmost importance when recovering, or restoring, the feasibility of the schedules. Generally, when disruptions occur, the solution to a number of highly interdependent problems is required quickly, e.g. crew and rolling stock changes must be coordinated with any timetable changes, and the planned depot schedules must also be revised. Due to the size and complexity of each of these problems, not to mention the short reaction time available to find a solution, it is not surprising that recently there has been an increased interest in computer aided decision support, or disruption management, tools that assist planners in recovery operations within the railway industry, see e.g. Kroon and Huisman (2011).

E-mail: {jhaa}{rmlu}{jesla}{dapi}@dtu.dk
Department of Engineering Management, Technical University of Denmark, Produktionstorvet, Building 426, 2800 Kgs. Lyngby, Denmark

In this work we consider two important rescheduling problems. In particular, we focus on the Rolling Stock Rescheduling Problem (RSRP) and the Depot Replanning Problem (DRP), respectively. We present an optimization-based framework, which extends the work of Haahr et al (2014), for simultaneously solving both of these in a disrupted environment. Rolling stock rescheduling necessitates reallocating train units to the timetabled trips in such a way that the overall cost of executing the revised schedule is minimized, while at the same time ensuring a sufficient number of seats are provided for the passengers. Additional constraints limiting the movement of available units, i.e. target inventory levels for each of the unit types at each of the depots, must also be respected. Depot planning, also known as the Train Unit Shunting Problem (TUSP) (see Freling et al (2005)) on the other hand, primarily focuses on the train units not in service. In order to cover the expected passenger demand as closely as possible, the composition of trains can be changed over the course of the planning horizon by coupling or decoupling units. Furthermore, units must routinely be taken out of service for maintenance and cleaning purposes. Depots usually consist of a number of parallel storage tracks where unavailable/unused units can be parked. On which track and in which order to park such units is the solution to the DRP. Often depot tracks can only be accessed from one end, implying that they function as *Last-in-First-out* (LIFO) stacks. However, even in the case of *free tracks*, i.e. tracks that can be accessed from both ends, ordering restrictions will stil apply. In addition each depot track has a certain capacity, limiting the number of units that can be assigned the track at any one time.

To the best of our knowledge, an approach integrating these two problems has not been previously proposed. The strategic and tactical level planning variants of each problem have been studied independently in the literature (see e.g Fioole et al (2006), Kroon et al (2008), and Freling et al (2005)), while models and methods are also available for the rolling stock rescheduling problem (see e.g Haahr et al (2014), Nielsen (2011)). However, research that explicitly addresses depot replanning, not to mention its integration with rolling stock rescheduling in a disrupted environment seems to be noticeably absent. Planning the rolling stock and depot movements sequentially in separate phases often leads to infeasible or sub-optimal solutions, while neglecting the depot planning aspect when rescheduling rolling stock units can result in infeasibilities with respect to some depots. A depot infeasibility would most likely be due to an ordering violation on one of the depot's tracks; i.e. two units cannot enter and exit the depot in the order prescribed by the rolling stock plan. Often such infeasibilities manifest themselves in networks with scarce depot capacity.

The method proposed in this paper extends the Branch-and-Price (BAP) framework of Haahr et al (2014), which reschedules disrupted rolling stock units, to also include the identification of parking plans for each of the depots in the network. In Haahr et al (2014) depot capacity is only implicitly considered in the form of an aggregated constraint on the total length of track available. It *does not* consider any ordering conflicts that could arise, nor if it is possible to park any unused units when individual track capacities are considered. It is a unit based decomposition which generates trajectories for each of the units in the fleet and in doing so identifies compositions for each of the trains that will be run. The underlying model ensures when coupling or decoupling units at a given depot, the resulting composition change is possible. From the specific individual unit trajec-

tories it is also possible to determine at which times a given unit enters, or exits, a given depot. Utilizing this information, we extend the BAP method of Haahr et al (2014) to a Branch-and-Price-and-Cut (BAPC) approach, where on finding a feasible solution to the routing problem, we test the feasibility of all depots in the network. If feasible, the solution is accepted, otherwise it is cut away and a new routing solution is generated. As such this paper makes the following important contributions to literature on railway rolling stock rescheduling:

- An integrated framework for recovering rolling stock and depot plans
- A comparison of methods for solving the Train Unit Parking Problem (TUP).
- A counter example highlighting the heuristic nature of a previously proposed optimal method in Freling et al (2005) for generating the optimal Linear Programming (LP) solution to the so-called TUP
- A heuristic swapping routine that can be used to repair a depot plan as an alternative to cutting the infeasible solution away
- An approach that removes one phase of planning usually required in the TUSP

This research is carried out in collaboration with DSB S-tog, the suburban rail operator in the greater Copenhagen area. All depots on this network are LIFO stacks and depot space is at a premium. Finding a feasible rolling stock schedule is only part of them problem since there may not exist any feasible parking plan to facilitate the required shunting movements. We test the performance of the algorithm on several real-life case studies provided by S-tog.

This paper is structured as follows. In Section 2 we introduce a more formal description of each of the two problems. Section 3 then provides a brief summary of the relevant literature, where any differences to existing approaches are elaborated on. We introduce mathematical models for the respective problems in Section 4. Section 5 discusses solution methods for the developed, individual models and describes in detail the complete, integrated framework. It is in this section that we also provide a counter example to the proposed optimal approach of Freling et al (2005) for the TUSP. In Section 6 the performance of the proposed methodology is analysed on a set of realistic test instances. Finally, conclusions and future research directions are outlined in Section 7.

2 Problem Description

Since this problem deals with the integration of two rescheduling problems, we separate this section into two subsections and provide a detailed description of each in turn. Section 2.1 introduces the RSRP, while Section 2.2 is devoted to a discussion on the DRP.

2.1 Rolling Stock Scheduling

Given a revised timetable, as a result of a disruption, the rolling stock rescheduling problem entails reallocating a fleet of units, possibly of different types, to the revised set of timetabled trips in such a way that some objective is optimized. The trips each unit can be assigned to depend on the unit's current location and possibly its unit type. A unit's type specifies the characteristics of the unit,

i.e. it's length, operating cost, and capacity. Typically, one tries to reallocate the fleet in such a way that sufficient seat capacity is provided for the passengers; however, minimizing the number of additional cancellations and/or operating costs might also be a preferred objective. Unlike, rolling stock planning, many of the constraints are soft since it is not certain that they can be fulfilled when facing a disruption. The constraints can be violated, but doing so incurs a penalty. Common soft constraints include: covering trips, covering the required seat demand, and adhering to the end-of-day balance. The end-of-day balance guarantees that a certain number of units of each type are available in each of the depots at the end of the day. This ensures that the rolling stock operations the following day can start as planned. Hard constraints include the current location of the unit types, the maximum length of train compositions on trips, and upper bounds on the available depot capacities. The upper bound on a depot's capacity simply gives the total track length available there. Depots typically consist of multiple parallel tracks (of different lengths), and such information is lost when giving this total capacity. That is, the aggregated lengths ensure no depot capacity will ever be violated; however, this does not rule out the possibility for ordering violations on the depot tracks when considering all tracks as well as the exact timings of unit movements into and out of the depot.

2.2 Depot Replanning

As mentioned earlier, the TUSP focuses on a specific depot and tries to allocate unit movements in such a way that each unit movement into and out of the depot can be performed in a conflict free manner and that no depot track capacity is ever violated. Traditionally, this requires solving two smaller subproblems known as the *Unit Matching Problem* and the *TUP*. The first is an optimization problem which matches arrival events with departure events and assigns each a physical unit. This is required as the rolling stock allocation is typically anonymous. In other words, only the type of unit required on a certain trip is known, not which actual unit will perform the trip Once matched, exact movement times for each physical unit are known and, based on these, the parking problem attempts to park them on the depot tracks in such a way that no movements are in conflict. Depending on arrival and departure times, if parked on the same track, one unit can prevent another from exiting the depot.

Due to the unit based routing perspective of the rolling stock model we solve, we only concern ourselves with the TUP. This is because all arrival and departure times at each of the depots for each of the units are implicitly contained in the variables of the rolling stock model. Such variables provide information on the composition of each train assigned to each trip. Hence, the order the units will enter and leave a given depot is known, if being decoupled or coupled. What remains is to see if they can be parked in a conflict-free manner. The DRP can hence be stated as follows. Given a set of rescheduled rolling stock routes, can the resulting *events* be parked in feasibly in the depot. Here an event refers to a physical unit with known arrival and departure times, a given length, and a prescribed unit type. We argue that in recovery mode, it is sufficient to detect feasibility of the depots as the majority of the cost is incurred in the routing phase. Note, if feasible, depots can always be resolved independently to achieve a better parking. One key

difference between the planning and rescheduling phases when determining how to park the units is that in the latter an initial fleet position (on the depot tracks) must be adhered to.

Feasibility is unlikely to be a problem if the number of depot tracks is sufficiently large; however, for DSB S-tog, depot capacity is a scarce commodity, meaning this *must* be considered when routing rolling stock units. If infeasible, feasibility can be restored if it is possible to swap the routes of two events in conflict. Swapping can only be done if the units are considered interchangeable. Units are considered interchangeable if they are of the same unit type and have feasible maintenance levels. Maintenance checks on units at DSB-Stog are issued at certain times and routes are allocated to units whilst adhering to this. Thus, swapping units is not considered a general remedy to correct for feasibility. Note that in this work, we do not route units from platforms to depot tracks on a detailed infrastructure level. Though we assume this is done in a post-processing phase, the number of simultaneous events can be limited in the rolling stock model.

3 Literature

In this section we review relevant literature in the field of RSRP and the TUSP. For the former we also include in the survey models and methods used for solving the tactical level problem of rolling stock planning. For the latter, however, to our knowledge no studies dedicated to real-time disruption management of depots exist. The ordering restrictions inherent in the TUSP are not unique to passenger rail, nor trains for that matter. We draw analogies with bus depot planning and freight rail car classification.

Over the past few years several different models have been put forward for solving rolling stock problems, both from the planning and the rescheduling perspective. An approach for determining the minimum circulation of train units required to operate a timetable is described in Schrijver (1993). A more elaborate Mixed Integer Programming (MIP) model that can handle the combining and splitting trains is proposed in Fioole et al (2006). This model forms the basis of Nielsen et al (2012), where a rolling horizon framework is developed for solving the RSRP. These three approaches can all be classified as anonymous unit flow models. As such, one is unable to track individual unit movements. As an alternative, Haahr et al (2014) presents a path based formulation for the RSRP and solve this using BAP. Each path specifies a route for a particular unit type through the network. Train compositions are handled by the master problem, making sure that any composition changes are legal. Note that the model does not assign physical unit routes but sets of routes that are consistent with the unit types available. This methodology does therefore not deal with anonymous units and is equally applicable at the tactical planning level.

Not surprisingly, a number of studies concerning variants of the TUSP have also been conducted. The problem of dispatching trams from a storage yard is addressed by Winter and Zimmermann (2000). In order to achieve a departure order satisfying the scheduled demand, several shunting operations may be necessary. The authors describe combinatorial optimization models as well as both exact and heuristic approaches for solving the real time dispatch problem. Scheduling trams in the morning is also the topic of Blasum et al (1999). The authors prove the NP

completeness of the problem of finding as assignment of parked trams (of different types in stacks) to departure times without any additional shunting movements. A graph theoretical approach to shunting train units in a railway depot is adopted by Stefano and Koči (2004). The complexity of several subproblems is studied and the objective of all problems is to minimize the number of tracks needed to park the units. Three heuristic approaches to train shunting in a workshop area are described by Jacobsen and Pisinger (2011), while Føns (2006) considers different mathematical models and approaches for the TUSP. Both latter studies consider cases arising at the Copenhagen suburban railway operator, DSB S-tog.

The problem of dispatching buses from a bus terminal is the focus Gallo and Miele (2001). Initially the problem is formulated using the Quadratic Assignment model approach of Winter and Zimmermann (2000). In addition, the authors also describe a new model that takes into account the fact that the buses can have different lengths. The resulting model is shown to be well suited to decomposition, and hence the authors present a lagrangian decomposition based approach. Real-life instances from the Florence Public Transportation Company are studied. It is concluded that the algorithm can find good quality solutions at low computational cost. Other bus parking related research includes Hamdouni et al (2006) and Hamdouni et al (2007). The former considers identifying robust parking solutions and argues that one should generate solutions in which the bus lanes have at most two different types. Having fewer types on a lane reduces the potential for ordering (or crossing conflicts), possibly at the expense of wasting parking capacity. The latter introduces a benders decomposition approach for minimizing bus type mismatches between arrival and departure pairs. That is, in the problem considered it is possible to supply a bus of a different type to that which is demanded, but at a cost. Having as few bus types on a given parking lane is also a priority.

In Freling et al (2005), the TUSP is separated into a matching problem and a TUP, where the matching is solved first and then used as input to the TUP. The authors propose a MIP model for the matching problem, while the parking problem is solved using a column generation approach. Computational experiments focus on case studies in The Netherlands. It is not completely clear if a column generation approach is necessary for the instances considered. Furthermore, we argue, by way of a counter example that the dominance criteria stated is not exact, i.e. it can potentially remove the optimal solution.

A similar decomposition is also suggested by Lentink et al (2006). However, two additional steps focusing on the routing of the train units from their arrival platforms to their designated depot tracks are also included. Between the matching of arrivals with departures and the parking of the units, an estimate on the cost of routing the trains is calculated. These costs are then used when solving the parking problem. Upon parking the units, actual routes from the platform to the depot tracks are obtained. A heuristic that sequentially routes the units is devised, and the complete routing solution is improved using a 2-opt heuristic.

In Haijema et al (2006) the authors propose a dynamic programming based heuristic for solving the TUSP. A realistic test case from the railway station Zwolle in The Netherlands is used to test the developed methodology. The test case considers a 24 hour period during which 45 units arrive and 55 units depart. The depot has 19 tracks with a total length of 4000m. The simple heuristic is fast and flexible and produces promising results; however, only a single instance from real-life is considered.

The work of Kroon et al (2008) extends the work of Freling et al (2005) and presents a fully integrated model for solving both the matching problem as well as the parking problem. A large MIP model is proposed that attempts to minimize the number of split compositions and the number of different unit types simultaneously parked on the same depot track. The authors indicate how the model can be strengthened through the addition of clique inequalities and also how to model more practical restrictions. For example, a discussion on how to deal with trains composed of several train units as well as how to model depot tracks that can be approached from both sides is included. The concept of a virtual shunting track is introduced in order to help identify which tracks should be heterogeneous and which tracks should be homogeneous from a unit type perspective. Computational experiments focus on two Dutch stations and consider up to 125 train units of 12 different types that need to be parked.

An advanced planning tool for shunting operations is described in van Wezel and Riezebos (2011). Like almost all previous work on the TUSP, this approach first solves a matching problem that determines how arriving units are matched to departing units. This is done via a network flow algorithm. A k-shortest path algorithm is used to assign a track to each unit, while a modified version an undirected k-shortest path is used to route the units. Finally the approach also assigns drivers to shunt operations.

Aside from passenger railway operators, ordering problems on storage tracks are also highly prevalent in the freight rail industry. In order to reach their final destination freight rail cars are sorted at so-called *classification* yards where incoming trains are sorted into blocks of rail cars that share the same destination. The blocks are subsequently combined to form outbound trains. The order in which to process inbound trains, which blocks to build, which track to assign a block, and how to build outbound trains, are all decisions that need to be made. For a survey of shunting in the freight rail industry, the reader is referred to e.g. Gatto et al (2009), and Boysen et al (2012).

To summarize, when solving the TUSP one first typically solves a matching problem before solving a TUP (and possibly a routing) problem. This stems from the fact that rolling stock allocation is typically done via MIP models which generate anonymous unit routes. In what follows, we show the proposed methodology circumvents the need for the matching phase due to the specific unit route generation in the approach of Haahr et al (2014).

4 Models

This section focuses on modelling the TUP. We omit a discussion on how we model and solve the rolling stock rescheduling problem since we assume this to be available to us; we adopt the BAP approach of Haahr et al (2014).

There are various ways to model and solve the TUP, and in this section we introduce two such mathematical formulations. In particular, Section 4.1 describes a standard MIP formulation, which we solve with the commerical solver Gurobi. Additionally, Section 4.2 discusses a formulation to which column generation can be applied. For the latter we develop a full BAP framework. We term these two approaches the *compact formulation* and the *column generation approach*, respectively. Furthermore, we test the performance of these approaches in Section 6.

We begin by introducing notation consistent to both of the proposed models. When solving TUP for a given depot we assume that we have a set of events, $\mathcal{E}$. Such a set can be constructed from a feasible routing solution to the RSRP by observing when units are coupled and decoupled at the relevant depot. Associated with each event $e \in \mathcal{E}$ is information concerning the arrival e_{arr} and departure d_{arr} time of the unit, the length of the unit l_e, the unit's type, and any initial position information if the unit was initially parked in the depot. Typically, the initial position states not only the track, but the index of the unit in the stack of units initially parked on the track.

4.1 Compact Formulation

This first formulation is similar in structure to the MIP formulations given in Lentink et al (2006), and Kroon et al (2008) for the TUP. This approach focuses on the events and attempts to determine the best assignment of events to tracks. Note that we define the problem to have an objective function here, despite the fact we solve it as a feasibility problem. For the description of the model we prefer to give a general interpretation. In this approach binary variables x_{et} are introduced and indicate whether or not event $e \in \mathcal{E}$ is assigned track $t \in \mathcal{T}$. Since it may be impossible to park all events, a second set of binary variables y_e is used to indicate whether or not $e \in \mathcal{E}$ is left uncovered in the final solution. Track length restrictions must be observed whenever a unit arrives at the depot. For this purpose we introduce the set $\mathcal{A} := \{(a,t) \in \mathcal{E} \times \mathcal{T} | \text{a is an arrival event}\}$ and $\mathcal{E}_a$. The first contains pairs of events and tracks while the second gives the set of events present in the depot upon arrival of event $a \in \mathcal{E}$. Associated with each decision variable is a cost. We denote the cost of assigning event $e \in \mathcal{E}$ to depot track $t \in \mathcal{T}$ as c_{et}. A penalty cost of M is assigned to the y_e variables, reflecting the unattractiveness of uncovering an event. Any pair of events which cannot be parked on the same depot track for LIFO ordering reasons are said to be in conflict, and all such pairs are contained in the set $\mathcal{C}$. The full formulation can now be given.

$$\text{Minimize: } \sum_{e \in \mathcal{E}} \sum_{t \in \mathcal{T}} c_{et} x_{et} + M \sum_{e \in \mathcal{E}} y_e \tag{1}$$

$$\sum_{t \in \mathcal{T}} x_{et} + y_e = 1 \qquad \forall e \in \mathcal{E}, \tag{2}$$

$$\sum_{e \in \mathcal{E}_a} l_e x_{et} \leq L_t \qquad \forall (a,t) \in \mathcal{A}, \tag{3}$$

$$x_{et} + x_{e't} \leq 1 \qquad \forall (e, e') \in \mathcal{C}, t \in \mathcal{T}, \tag{4}$$

$$x_{et} \in \{0,1\} \qquad \forall e \in \mathcal{E}, t \in \mathcal{T}, \tag{5}$$

$$y_e \in \{0,1\} \qquad \forall e \in \mathcal{E}, \tag{6}$$

A brief description of the compact model is given as follows. The objective function (1) seeks a minimum cost assignment of events to depot tracks. Constraints (2) enforce the requirement that each event is assigned to exactly one of the depot tracks or is left uncovered Constraint set (3) ensures that the track

capacity is never violated when an arrival event occurs, while constraints (4) stipulate the LIFO restrictions; there is a pairwise packing constraint included for any two events in conflict. Finally, variable domains are given by (5) and (6). Note that constraints (4) can be strengthened by identifying stronger clique inequalities. Typically this set of constraints can be problematic as the formulation could become prohibitively large if there are many conflicts.

4.2 Column Generation Model

The second formulation we introduce is similar to that of the compact formulation above; however, constraints (3) and (4) are enforced in the construction of the variables and are thus not explicitly needed in the formulation. It has been proposed by, among others, Freling et al (2005) and Føns (2006). In this approach one looks at identifying *parking plans* for each of the tracks, and is the formulation obtained by applying Dantzig-Wolfe Decomposition to the MIP formulation above, where constraints (3) and (5) are placed in the subproblem and implicitly satisfied in the construction of the variables. A parking plan for a given track $t \in \mathcal{T}$ simply refers to a set of events that can all use track t in a conflict-free manner without violating the track capacity. As such, we denote the set of all parking plans for $t \in \mathcal{T}$ as $\mathcal{P}_t$ and introduce a new set of binary variables x_{tp} that govern the selection of a particular track plan $p \in \mathcal{P}_t$ for depot track $t \in \mathcal{T}$. A cost c_{tp} is associated with each such variable and indicates the unattractiveness of the assignment; this cost is merely the sum of the costs of each of the events contained in the parking plan. Again, we define y_e variables, one for each event, to include the possibility of not covering an event. Like, the compact formulation, each is assigned a large penalty cost M. Finally, the binary parameter a_{ep} is used to indicate whether event $e \in \mathcal{E}$ is included in track plan $p \in \mathcal{P}_t$ or not. The full formulation is given below.

$$\text{Minimize: } \sum_{t \in \mathcal{T}} \sum_{p \in \mathcal{P}_t} c_{tp} x_{tp} + M \sum_{e \in \mathcal{E}} y_e \tag{7}$$

$$\sum_{p \in \mathcal{P}_t} x_{tp} = 1 \qquad \forall t \in \mathcal{T}, \tag{8}$$

$$\sum_{t \in \mathcal{T}} \sum_{p \in \mathcal{P}_t} a_{ep} x_{tp} + y_e = 1 \qquad \forall e \in \mathcal{E}, \tag{9}$$

$$x_{tp} \in \{0,1\} \qquad \forall t \in \mathcal{T}, p \in \mathcal{P}_t, \tag{10}$$

$$y_e \in \{0,1\} \qquad \forall e \in \mathcal{E}. \tag{11}$$

An interpretation of this model is as follows. The objective function, given by (7), is identical in structure to that of (1); however, the aim here is to find a minimum cost selection of track plans. Constraints (8) ensure that each depot track is assigned exactly one of its possible track plans (including the null assignment, i.e. a parking plan containing no events), while constraints (9) enforce the restriction that each event must be present in exactly one of the track plans, or left uncovered. Variable domains are stated by (10) and (11). While this formulation has a constant row dimension (independent of the number of conflicting events), it can have an exponential number of variables; for large problems there could

be many possible parking plans. Hence, a column generation approach is typically the preferred solution method, generating only variables that have the potential to improve the objective function. This approach was first described in Freling et al (2005), where it is coupled with a heuristic Branch-and-Bound (BAB) strategy. In Section 5 we outline a full BAP procedure, and correct an error in the proposed column generation procedure.

5 Solution Approach

Solution methods for both of the formulations given in Section 4 are described in this section. For Model (1)-(5) one can observe that it can be potentially cumbersome to a priori generate all pairs of conflicts, constraints (3), a priori. Therefore, as well as solving the model directly with the commercial solver Gurobi, we also test an approach that solves a relaxed version of the model in which all such LIFO restrictions are initially removed and gradually reintroduced via a Branch-and-Cut (BAC) procedure if any are found to be violated. The second model, on the other hand, is characterized by exponentially many variables (parking plans) that must be considered. For this model, we describe the column generation model proposed in Freling et al (2005) and show how, through constraint branching, this can be extended to a full BAP framework. More details regarding each of the approaches is described in the following sections.

5.1 Branch-and-Cut

BAC is a well known technique for efficiently solving large scale mixed MIPs, see e.g. Desrosiers and Lübbecke (2010). This approach combines the addition of cutting planes within a BAB framework. More specifically, whenever the relaxation of node in the branch-and-bound tree is solved to optimality, so-called *separation* routines are performed. Each separation routine attempts to identify valid inequalities, i.e. constraints that are not part of the original formulation, but which are violated by the node's fractional solution. By adding such valid inequalities one hopes to improve the node's objective bound by removing infeasible fractional solutions that would otherwise be branched on. The valid inequalities produce a tighter relaxation and often a less fractional solution.

In addition to generating valid inequalities from the problem's full constraint set it is also possible to remove a set of problem constraints, thus obtaining a relaxation, and run a separation routine that identifies any violated, relaxed constraints. This can be particularly useful if there is a large set of constraints, many of which are unlikely to be binding in an optimal solution. See, for example, the application of BAC algorithms to the travelling salesman problem and its variants, see e.g. Padberg et al (1987), where the exponential number of sub tour elimination constraints can be more efficiently handled via a separation routine.

In addition to a direct solve of Model (1)-(6), we also compare an alternative BAC algorithm for the TUP. As stated above, we relax the problem by first removing the LIFO related constraints (4), and whenever a violation is found, the necessary constraints are added. Note that if a LIFO violation is found for two events, we add a LIFO constraint for each track in the depot to ensure that the

conflict doesn't simply move to one of the other tracks. In this approach we cut on both fractional and integer solutions.

5.2 Column Generation

Column generation is an efficient method for solving large scale linear programs and (see e.g. Desrosiers et al (2005)), via a BAP algorithm (see e.g. Barnhart et al (1998)), can be adapted to solve large scale mixed integer programs. It is typically used in situations where it is computationally intractable to enumerate all possible variables a priori. With column generation, the problem is decomposed into a *master* problem and one or more independent *subproblems*. The role of the master problem is to solve a reduced set of the possible variables to (linear programming) optimality, while the role of each subproblem is to generate promising variables for the master problem using the dual variables from the optimized master solution, much like the pricing step in the simplex algorithm. Column generation is an iterative procedure between the master problem and subproblems and continues until no variables with negative reduced cost are found by any of the subproblems.

As shown in Freling et al (2005), the TUP lends itself very naturally to a column generation framework; the relaxed, restricted master problem is obtained by relaxing the integrality restrictions on the x variables and restricting the set of parking plans in the model, i.e. the master problem is given by (7)-(9), (11), and $x_{tp} \geq 0$. We introduce dual variables π_t for each $t \in \mathcal{T}$ and μ_e for each $e \in \mathcal{E}$. The former are associated with constraints (8), while the latter correspond to constraints (9). A subproblem can then be identified for each depot track, where the aim is to find an improving parking plan given the optimal solution to the relaxed, restricted master problem. Such an approach allows one to only generate those parking plans that have the potential to improve the objective function and avoid considering many variables that will assume a non-basic status in the optimal linear programming solution. As Freling et al (2005) points out, the problem of generating favourable reduced cost parking plans can be modeled as a Resource Constrainted Shortest Path (RCSP). We provide a brief overview of this in Section 5.2.1. In particular, we show how the methodology is applied to depot tracks that function as LIFO stacks. This overview is necessary in order to be able to follow the dominance counter example we provide in Section 5.2.2.

5.2.1 Resource Constrained Shortest Path

RCSP problems typically arise in the application of column generation to vehicle routing and crew rostering problems. The problem entails finding a shortest path between two nodes of a network, while adhering to a number of so-called *resource constraints*. The network is graphical representation of the problem at hand, while the resource constraints impose restrictions on the solution to the subproblem. To model the TUP, Freling et al (2005) introduce an acyclic layered network. In addition to a source node, $\mathcal{O}$ and sink node $\mathcal{D}$, the node set contains two nodes for each event $e \in \mathcal{E}$ for LIFO tracks. The first indicates the event is parked on the track and other indicates the opposite. The nodes for each event constitute a layer, and these layers are sorted by the arrival time of the corresponding events. Arcs are used to connect the nodes of one layer with nodes in the subsequent layer.

The source node is connected to nodes in the first layer, while the nodes of the last layer are connected to the sink. Figure 1 gives an example of such a network for the TUP with five events. The arcs represent parking choices: whether to park an event on the track or not. The cost of an arc is the reduced cost contribution of the event. In other words, it is the cost of parking the event c_{et} minus the dual value on the constraint it covers, μ_e. For convenience, π_t is included on the first event. Rectangles are used to indicate layers.

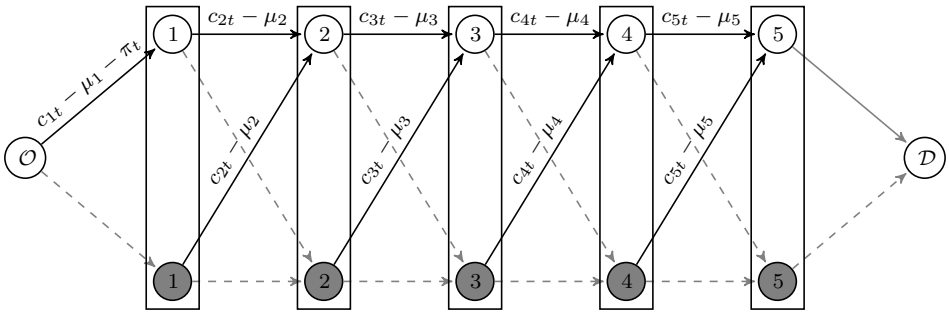

Fig. 1 An example subproblem network with five events. Every event constitutes two nodes that indicate whether the event is parked or not on the track. All paths originating in $\mathcal{O}$ and terminating in $\mathcal{D}$ represent a plan, which is feasible if the resource constraints are respected.

It is clear to see that enumerating all paths in this network would provide all subsets of the considered events. However, not all subsets are feasible. The resource constraints that must be observed are the remaining length of the depot track being considered and the minimum departure time of events parked on the track in a partial path. The first ensures track capacity is never violated, while the second prevents conflicting movements from being parked on the track. RCSP problems are typically solved using a label setting algorithm in which the nodes are considered in topological order. A label is associated with a particular node and contains information about the partial path to that point, i.e. the cost and the values each of the resource levels. New labels are generated by *extending* a label at a given node to each of the node's successor nodes. The new label reflects the label from which it is generated, updated with any changes to resource levels that have occurred in transitioning the arc. Dominance strategies are used to restrict the number of labels generated. For an introduction on solving RCSPs, the reader is referred to Irnich and Desaulniers (2005) and Irnich (2008).

5.2.2 Dominance Rule Counter Example

The proposed solver for the subproblem described in section 5.2.1 is similar to the Dynamic Program Algorithm described and tested by Freling et al (2005). Their algorithm is more general, as it can handle free tracks. We should therefore also be able to adopt their algorithm for our problem, however, we have identified an error in their algorithm.

The proposed domination rule presented by Freling et al (2005) is in fact incorrect. Granted, the overall column generation framework in the paper is a

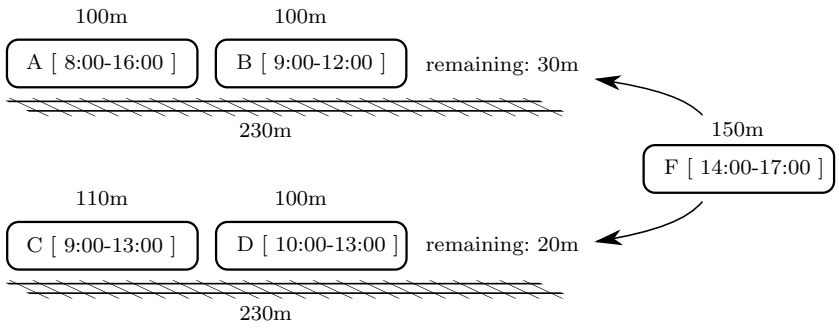

Fig. 2 Example of two partial parking plans. The boxes represent train units of individual lengths and parking durations. Both plans do not park event E, leaving them in the same state (node in the graph), thus making it possible to check for dominance. The lower pattern can not be dominated since the upper pattern cannot include event F in contrast to the lower pattern. If F has a negative cost, then a feasible and better pattern is wrongly pruned.

heuristic as columns are solely generated in the root node. However, the described algorithm for the subproblem does not guarantee optimal solutions because the described domination rule is faulty. We prove this by issuing a counter-example where the algorithm fails to find an optimal solution. We note that the solutions produced by the incorrect algorithm are always feasible, which essentially makes the algorithm a heuristic approach. Freling et al (2005) state the following in section 5.2.2:

> A path $p_i \in P_u$ is dominated by a path $p_j \in P_u$ if all the resource variables of p_i are dominated by those of p_j

This is a standard domination rule that is valid for *regular* resource consumption problem. Parking on tracks is however a somewhat temporary resource consumption as it occupies the tracks for a limited time, and blocks units behind it during this period. The resources in question are the total cost, the remaining length of a track (path) and the "earliest departure time of the blocks in path p, which had not left as of the time of node u", p269. In the case of a LIFO track the last resource is the departure time of the *outmost* rolling stock, i.e., the unit which is blocking all the other ones on the track.

Our counter-example is illustrated in Figure 2. In the example a 230 meter track is considered, together with events $\{A, B, C, D, E, F\}$, arriving in the listed (alphabetical) order. Assume the costs of the events to be $\{-2, -2, -1, -2, 10, -10\}$, respectively. Consider the case where the solver is testing whether one label (i.e. parking plan) dominates the other one. Assuming that unit E arrives before either B or D have left, the current state of each label remains the same when processing the node indicating that E will not be parked (see Figure 1). At this point the top label and bottom labels have resource consumption vectors $(-4, 30, 12:00)$ and $(-3, 20, 13:00)$. It is clear that by definition the bottom label is dominated because it has a higher cost, less remaining track capacity and the earliest departure time is greater. However, the optimal solution $\{C, D, F\}$ with cost -13 will not be found, as the prefix $\{C, D\}$ has been pruned.

In our preliminary results we identified cases where the non-optimal dominance actually results in non-optimal solutions. As expected, the simpler non-optimal

dominance results in faster solution times. The method could therefore be used as an heuristic to speed up overall convergence in a BAP framework. Hence, we revise the dominance criteria. In our approach, domination between labels only occurs, if in addition to the remaining length and cost criteria, all units currently parked on the track in one label have an earlier (or identical) departure time than those of the other label.

5.2.3 Constraint Branching

To ensure optimality of the column generation framework, we utilise the well known constraint branching technique, developed by Ryan and Foster (1981), for solving set partitioning problems. In an optimal, but fractional solution to such a problem there exists two constraints (say c_1 and c_2) such that $\sum_{j \in J(c_1,c_2)} x_j < 1$, where $J(c_1, c_2)$ defines the set of variables that cover both constraint c_1 and c_2. Two branches are then created by ensuring that either $\sum_{j \in J(c_1,c_2)} x_j \geq 1$ (*one* branch), or $\sum_{j \in J(c_1,c_2)} x_j \leq 1$ (*zero* branch). On the one branch both constraints must be covered by the same variable, while in the zero branch, they must be covered by different variables. For the TUP, we branch on track and event pairs. In other words, on the one branch, an event is forced onto a depot track, while on the zero branch it is prohibited from using a given depot track. In an optimal, fractional solution, the track and event pair with the closest fractional coverage to 0.5 is selected to branch on.

When branching, all variables that are in the master problem and which do not satisfy the imposed branches are removed, and the subproblems are modified to ensure they return parking plans consistent with the imposed branches. One advantage of this constraint branching approach is that minimal changes to the subproblems are required. Except form removing selected columns the master problem remains unchanged. When branching, we adopt a depth first strategy in which we successively enforce *one* branches.

5.3 Unit Swapping

If not all units can be parked in a solution to the TUP, before rejecting the unit routing solution we attempt to swap "interchangeable" units. We consider two units to be interchangeable if they are of the same unit type and if, after swapping, the resulting routes can be performed feasibly (i.e. without violating maintenance levels). A swap results in an exchange of the departure times of the two events in question. To swap units, we propose a heuristic that iterates over the set of unparked units, $\mathcal{U}$ and, for each, identifies a set of swapping possibilities $\mathcal{S}_u$ by considering the track assignment of each of the parked events in the depot (at the unparked unit's scheduled arrival time) as well as the next units to arrive; i.e. we attempt to swap an unparked unit with an already parked unit (one at the front of a stack) or with one of the next units to arrive, using the solution to the TUP. Algorithm 1 gives an overview of the procedure.

The algorithm terminates when no unparked events exist, no swapping possibilities remain, or an upper limit on iterations has been reached. We allow one swap per track at each iteration. Swap possibilities are ranked such that a unit with

Algorithm 1 Unit Swapping Heuristic Heuristic

```
1: procedure UnitSwap(sol)
2:     iterations ← 0
3:     swappable ← true
4:     while unparked units exist and swappable and iterations < limit do
5:         U ← getUnparkedUnits(sol);
6:         S ← ∅
7:         for u ∈ U do
8:             S_u ← ∅
9:             for t ∈ T do
10:                S_ut ← createSwapPossibilities(sol, t, u, u_arr)
11:                S_u ← S_u ⋃ S_ut
12:            S ← S ⋃ S_u
13:        if swapping possibilities exist then
14:            rankSwaps(S)
15:            performSwaps(S)
16:            sol ← resolveDepot()
17:        else
18:            swappable ← false
```

fewest swaps is considered first. To ascertain the impact of the chosen swaps, the depot is resolved. The algorithm's sequential nature makes it inherently heuristic.

5.4 Integrated Rolling Stock and Depot Framework

In the following we propose a framework that solves the Depot Problem and Rolling Stock Scheduling simultaneously in an integrated loop. The main assumption in this framework is that the depot problem is a feasibility problem, i.e., only the rolling stock schedule contributes to the objective and that no cost stems from the depot problem. An overview of the framework in shown in Figure 3.

The first component is an exact method for determining an optimal rolling stock schedule. We adopt the BAP method of Haahr et al (2014) to solve the problem. In principal, any method can be used that fulfills a few criteria. In this integrated loop the method must provide the activities for each individual rolling stock as well as the compositions formed on each train trip service such that events can be identified.

The second component is the Depot Movement Planner that solves the problem presented earlier in Section 2. In this paper we presented two different approaches for solving this problem, c.f. Section 4. Either one can be used interchangeably but during testing we will adopt the MIP based method.Using the found rolling stock schedule we derive all arrival and departure events by inspecting coupling- and decoupling performed in the schedule. The scheduled events and the predefined initial fleet positions define the input to the Depot Movement Planner. A globally feasible and optimal solution is found if a depot movement plan can be found for each individual depot.

If no feasible depot movement plan exists for the provided rolling stock schedule we can conclude that the specified events are unparkable. However, there may exist multiple rolling stock route assignments for obtaining the same composition assignment of the train trips. Note, that an underlying assumption is that cost of a rolling stock schedule is entirely determined by the composition assignment. Thus,

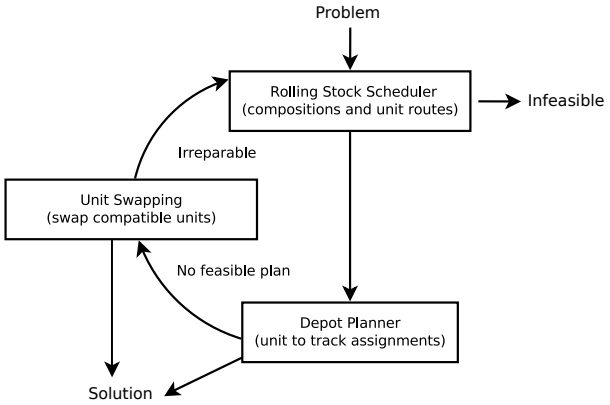

Fig. 3 Diagram illustrates the flow in the integrated framework. The framework consists of three major components connected in a loop. In the loop a rolling stock schedule is found first. This schedule is fed to the depot planner. The loop terminates if a feasible depot plan exists, otherwise the framework tries to swap units in the depot. If no suitable swapping can be found, then the rolling stock assignment is rejected, and another one must be found.

if a no valid parking plan exists for the current rolling stock routes, then there may still exist another set of routes which is feasible. Since enumerating all possible set of routes is intractable we instead propose a method to identify potential swapping points for the current routes. The method is described in Section 5.3.

If a feasible unit swapping can be obtained, such that units can be parked, we have found a globally feasible and optimal solution. Note, that unless we apply an exact method for swapping units, the overall is not guaranteed to be optimal. In this paper we adopt a heuristic method.

Finally, if no feasible depot movement parking can be found in the unit swapping method the framework rejects the rolling stock composition. A cut is generated, based on the infeasible depot, such that the combination of composition trip assignments, either originating or terminating at this depot, are prohibited. Resolving the rolling stock schedule with the added constraint will therefore find a new schedule and a new iteration in the loop is initiated.

6 Computational Results

To determine the best way of solving the TUP we benchmark the approaches described in Section 6 on a test set of 11 artificial data sets. The TUP is a subcomponent of a larger framework and it is critical that this is efficient as possible. We provide an overview of this benchmarking in Section 6.1. Results of the integrated framework on DSB-Stog case studies are given in Section 6.2.

6.1 TUP Benchmark

We begin with a short description of the artificial data sets. Artificial data sets have been chosen as the provide the flexibility to vary the level of, among other

things, the level of infeasibility, the number of events and depot tracks, as well as the number of unit types. Table 1 states the most important characteristics of each instance. For each instance the following information is provided: the number of events, the number of depot tracks, the length of the longest track (denoted L_{max}), the planning horizon, the number of unit types, and the unit type lengths. Note that the instances have been purposely created with infeasibilities, and that these are in line with the size of the problems DSB-Stog faces.

Instance	$\|\mathcal{E}\|$	$\|\mathcal{T}\|$	L_{max}	Horizon (s)	Types	Unit Lengths
data0	66	6	300.0	17113	2	[35,70]
data1	69	6	500.0	17785	2	[35,70]
data2	62	5	850.0	21103	2	[42,84]
data3	75	5	850.0	24837	3	[30,60,90]
data4	72	5	700.0	21602	2	[42,84]
data5	59	5	740.0	21602	3	[30,60,90]
data6	79	5	800.0	25202	2	[35,70]
data7	79	6	790.0	25202	3	[35,50,75]
data8	78	7	900.0	24897	1	[42]
data9	101	8	1000.0	24964	2	[42,84]
data10	109	8	400.0	28529	2	[42,84]

Table 1 Test instances

		BAP Framework				BAC_1	BAC_2		*FEL*	
Instance	Z^*	*cols*	n	l	t (s)	t (s)	t (s)	%	t (s)	Z
data0	8	1030	37	18	1.33	3.21	*	12.50	1.30	11
data1	6	1317	41	20	1.74	0.36	*	16.67	1.43	7
data2	7	727	13	6	0.75	0.12	5.05	0.00	0.41	8
data3	8	1181	23	11	1.71	0.11	7.63	0.00	2.38	10
data4	11	834	25	12	1.05	0.14	*	9.09	1.17	14
data5	4	754	29	14	0.98	0.05	1.09	0.00	0.43	8
data6	11	841	17	8	1.18	1.99	*	36.36	0.58	14
data7	8	1740	41	20	2.18	1.76	*	12.50	1.83	9
data8	1	3236	99	49	4.86	0.05	0.36	0.00	3.78	1
data9	3	5125	87	43	17.33	0.42	*	66.67	6.78	3
data10	0	2413	147	73	6.14	0.23	1.29	0.00	5.17	0

Table 2 Method Comparison

We solve each of the instances in Table 1 using four different methods: our BAP approach, two BAC approaches, and the column generation approach of Freling et al (2005) combined with constraint branching. BAC_1 is a direct solve of Model (1)-(6) (with default Gurobi cut generation), while BAC_2 also separates violated LIFO cuts. For our BAP approach we report the number of columns generated (*cols*), the number of nodes in the BAP tree (n), the deepest level of same tree (l) and the time needed to solve the instance (t). For the two BAC approaches we give the time and, for BAC_2, the gap from optimality at termination. For Freling et al (2005), we give the solution time along with the objective value obtained, Z. For every instance we give Z^*, the optimal solution, which counts the number of unparked units. The results are given in Table 2. All tests have been performed

on Intel(R) Xeon(R) CPU X5550 @ 2.67GHz with 24GB ram running Ubuntu Linux 14.04. The commercial solver Gurobi 6.0 is used for both BAC approaches, while Cplex 12.61 is used to solve the LP relaxations of the other two methods. An upper bound of 10 minutes is enforced on the solution time. From the results it can be seen that BAC_1 is the most superior, and that the approach of Freling et al (2005), while faster than the BAP, provides suboptimal solutions.

In addition we also report the results of the swapping heuristic; these are given in Table 3. We compare the BAP approach with BAC_1. For each we report the new objective (Z), the number of swaps, and the time taken. For any instance, up to 30 swapping iterations are allowed. In almost all cases feasibility through swapping can be achieved, and the heuristic nature is seen by the fact the two approaches yield different results at times. Currently the number of swaps is not optimized; however, this "re-matching" problem could also be performed via a MIP if running times are fast enough.

		Column Generation			BAC_1		
Instance	Z^*	Z	*Swaps*	t (s)	Z	*Swaps*	t (s)
data0	8	0	12	7.383	0	19	3.39
data1	6	0	10	20.35	0	11	4.31
data2	7	0	11	5.50	0	16	1.33
data3	8	0	12	15.08	3	15	2.90
data4	11	1	21	126.10	1	16	5.12
data5	4	0	4	3.04	0	5	0.21
data6	11	0	22	18.748	1	18	15.58
data7	8	1	16	145.32	0	18	44.17
data8	1	0	1	8.70	0	1	0.14
data9	3	0	4	42.27	0	4	1.23

Table 3 Swap Results

6.2 Integrated Rolling Stock and Parking

In this section we perform test on the integrated framework described in section 5.4. We use a real-life timetable instances from the Suburban Railway Operators in Copenhagen (S-tog). They operate a weekly schedule that consists of four daily distinct timetables. The main distinction is between weekdays and the weekend, but Friday and Saturday also include additional night train services. The instances are shown in Table 4. The first results are listed in Table 5. Here we have adopted a balanced set of penalties for seat-shortages, driven mileage, end-of-day balance deviations, couplings and de-couplings, see Haahr et al (2015). These initial results demonstrate runtimes which are acceptable for planning purposes. However they also show that the all found rolling stock schedules are feasible, thus the integrated loop only executed the first iteration.

The next benchmark generates schedules with more depot activity, thereby potentially increasing the difficulty of the depot problems. For each instance we gradually reduce the penalty of performing mid-trip couplings, i.e., couplings or decouplings that do not occur at the beginning or end of a trip sequence. Trip

Name	Stops	Trips	Trips*	Weekday	Lines
Fri	28 719	4 558	886	Friday	A,B,Bx,C,E,F&H
Sat	20 474	1 916	590	Saturday	A,B,C&F
Sun	19 919	1 871	574	Sunday	A,B,C&F
Mon	28 017	4 468	868	Monday	A,B,Bx,C,E,F&H

Table 4 Four timetables operated by DSB S-tog. The columns respectively show the instance names, total number of stops, total number of trips, total number of non-reducible trips (Trips*), weekday, and finally the lines that are running.

						Depot	
Instance	Time (s)	RS	Tree	Columns	Solutions	Feasible	Infeasible
Fri	79	74%	14	4477.0	4.0	4.0	0
Mon	96	83%	7	5828.0	1.0	1.0	0
Sat	16	61%	9	1495.0	7.0	7.0	0
Sun	10	63%	5	1392.0	1.0	1.0	0

Table 5 Results when running integrated framework with previously tested penalties. The columns respectively show instance name, runtime, percentage of runtime (in seconds) spent by the rolling stock framework, number of processed nodes in the RS BAB tree, number of generated columns, number of RS solutions founds, number of feasible RS solutions, and finally number of infeasible RS solutions.

sequences usually span most of the day, shuttling between the same two terminal stations. The results are shown in Table 6. We observe an expected increase in the number of couplings as the penalty reduces, however this does not seem to affect the feasibility of the corresponding depot problems.

7 Conclusions

We present and benchmark four different approaches for solving the depot parking problem. Judging by the results there is no reason to favour the column generation framework over the MIP based approach. However, it would be interesting to see if one method scales better than the other with larger instances. Separating violated LIFO constraints also seems tedious. A direct solve of the full MIP gave the best results, solving each artificial instance within 4 seconds. These instances do represent the largest depot facilities at DSB S-Stog. Due to the additional complexity of implementing a column generation approach we deem it appropriate for future research to consider a MIP based solution approach first. The complexity is apparent since even formulating a correct domination criteria can include some pitfalls. We have shown how a previously proposed approach is non-optimal, see Freling et al (2005). In some cases there were significant differences between our BAP framework, and the method of Freling et al (2005). Furthermore, run times produced by the integrated framework are acceptable.

Rolling stock scheduling and depot parking planning have traditionally been solved sequentially in isolation. Planners at S-tog have identified this sequential approach problematic, as it is sometimes impossible to find a feasible parking plan for initially found rolling stock schedules. Our tests did however surprisingly reveal no difficulties in parking initially found rolling stock schedules. Results are found

Instance	Cost	Couplings	Time (s)	RS	Solutions	Depot	
						Feasible	Infeasible
Fri	600	48	128	82%	1	1	0
	700	41	154	78%	1	1	0
	800	34	80	83%	1	1	0
	900	30	94	77%	1	1	0
	1 000	27	79	76%	4	4	0
Mon	600	45	218	82%	1	1	0
	700	38	133	79%	4	4	0
	800	32	116	84%	1	1	0
	900	28	115	83%	1	1	0
	1 000	26	111	84%	1	1	0
Sat	600	29	23	71%	1	1	0
	700	25	18	68%	1	1	0
	800	23	23	67%	1	1	0
	900	21	13	64%	1	1	0
	1 000	17	11	61%	7	7	0
Sun	600	37	23	70%	1	1	0
	700	31	17	71%	1	1	0
	800	28	12	68%	1	1	0
	900	26	13	66%	1	1	0
	1 000	22	9	63%	1	1	0

Table 6 Results when running integrated framework with different coupling penalties. The columns respectively show instance name, coupling penalty, number of mid-trip couplings, runtime, percentage of runtime (in seconds) spent by the rolling stock framework, number of RS solutions founds, number of feasible RS solutions, and finally number of infeasible RS solutions.

in reasonable time for planning purposes, and in future work we will investigate other aspects of the parking problem which may lead to infeasibility in real-life.

In future work we also want to investigate alternative integrated frameworks for finding rolling stock schedules and depot parking plans simultaneously. The main drawback of the current framework is the high complexity of performing swaps optimally without enumerating too many rolling stock unit paths.

Previous work in literature includes some modelling choices with respect to train compositions and marshalling. This addition has been omitted in this work and seems like an obvious choice for future research.

References

Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MWP, Vance PH (1998) Branch-and-price: Column generation for solving huge integer programs. Operations Research 46(3):316–329

Blasum U, Bussieck MR, Hochstättler W, Moll C, Scheel HH, Winter T (1999) Scheduling trams in the morning. Mathematical Methods of Operations Research 49(1):137–148

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

Desrosiers J, Lübbecke ME (2010) Branch-price-and-cut algorithms

Desrosiers J, Lübbecke M, Solomon MM (2005) Column generation. In: Desaulniers G, Desrosiers J, Solomon MM (eds) A Primer in Column Generation, Springer: New York, chap 1, pp 1–32

Fioole P, Kroon LG, Maróti G, Schrijver A (2006) A rolling stock circulation model for combining and splitting of passenger trains. European Journal of Operational Research 174(2):1281–1297

Føns P (2006) Desicion support for depot planning in the railway industry. Master's thesis, Technical University of Denmark

Freling R, Lentink RM, Kroon LG, Huisman D (2005) Shunting of passenger train units in a railway station. Transportation Science 39(2):pp. 261–272

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

Gatto M, Maue J, Mihalák M, Widmayer P (2009) Shunting for dummies: An introductory algorithmic survey. In: Ahuja RK, Möhring RH, Zaroliagis CD (eds) Robust and Online Large-Scale Optimization, Lecture Notes in Computer Science, vol 5868, Springer Berlin Heidelberg, pp 310–337

Haahr J, Lusby R, Larsen J, Pisinger D (2014) A Branch-and-Price Framework for Railway Rolling Stock Rescheduling During Disruptions. DTU Management Engineering

Haahr JT, Kroon LG, Veelenturf LP, Wagenaar JC (2015) Comparing two exact methods for passenger railway rolling stock scheduling. In: Hansen I (ed) Proc. 6th International Conference on Railway Operations Modelling and Analysis (RailTokyo2015)

Haijema R, Duin C, van Dijk NM (2006) Train Shunting: A Practical Heuristic Inspired by Dynamic Programming, John Wiley & Sons, Inc., pp 437–475

Hamdouni M, Desaulniers G, Marcotte O, Soumis F, van Putten M (2006) Dispatching buses in a depot using block patterns. Transportation Science 40(3):364–377

Hamdouni M, Desaulniers G, Soumis F (2007) Parking buses in a depot using block patterns: A benders decomposition approach for minimizing type mismatches. Comput Oper Res 34(11):3362–3379

Irnich S (2008) Resource extension functions: properties, inversion, and generalization to segments. OR Spectrum 30(1):113–148

Irnich S, Desaulniers G (2005) Shortest path problems with resource constraints. In: Desaulniers G, Desrosiers J, Solomon M (eds) Column Generation, Springer US, pp 33–65

Jacobsen PM, Pisinger D (2011) Train shunting at a workshop area. Flexible Services and Manufacturing Journal 23(2):156–180

Kroon L, Huisman D (2011) Algorithmic support for railway disruption management. In: Nunen JA, Huijbregts P, Rietveld P (eds) Transitions Towards Sustainable Mobility, Springer Berlin Heidelberg, pp 193–210

Kroon LG, Lentink RM, Schrijver A (2008) Shunting of passenger train units: An integrated approach. Transportation Science 42(4):436–449

Lentink RM, Fioole PJ, Kroon LG, van't Woudt C (2006) Applying Operations Research Techniques to Planning of Train Shunting, John Wiley & Sons, Inc., pp 415–436

Nielsen LK (2011) Rolling stock rescheduling in passenger railways. PhD thesis, Erasmus University Rotterdam

Nielsen LK, Kroon L, Maróti G (2012) A rolling horizon approach for disruption management of railway rolling stock. European Journal of Operational Research 220(2):496–509

Padberg M, , Rinaldi G (1987) Optimization of a 532-city symmetric traveling salesman problem by branch and cut. Operations Research Letters 6(1):1–7

Ryan DM, Foster BA (1981) An integer programming approach to scheduling. In: Wren A (ed) Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling, North-Holland, pp 269–280

Schrijver A (1993) Minimum circulation of railway stock. CWI QUARTERLY 6:205–217

Stefano GD, Koči ML (2004) A graph theoretical approach to the shunting problem. Electronic Notes in Theoretical Computer Science 92(0):16 – 33, proceedings of ATMOS Workshop 2003

van Wezel W, Riezebos J (2011) Case study: Advanced decision support for train shunting scheduling. In: Fransoo JC, Waefler T, Wilson JR (eds) Behavioral Operations in Planning and Scheduling, Springer Berlin Heidelberg, pp 413–430

Winter T, Zimmermann UT (2000) Real-time dispatch of trams in storage yards. Annals of Operations Research 96:287–315