Assignment Based Resource Constrained Path Generation for Railway Rolling Stock Optimization

Boris Grimm
Freie Universtät Berlin, Germany
Zuse Institute Berlin, Germany

Ralf Borndörfer
Freie Universtät Berlin, Germany
Zuse Institute Berlin, Germany

Julian Bushe
Zuse Institute Berlin, Germany

Abstract

The fundamental task of every passenger railway operator is to offer an attractive railway timetable to the passengers while operating it as cost efficiently as possible. The available rolling stock has to be assigned to trips so that all trips are operated, operational requirements are satisfied, and the operating costs are minimum. This so-called Rolling Stock Rotation Problem (RSRP) is well studied in the literature. In this paper we consider an acyclic version of the RSRP that includes vehicle maintenance. As the latter is an important aspect, maintenance services have to be planned simultaneously to ensure the rotation's feasibility in practice. Indeed, regular maintenance is important for the safety and reliability of the rolling stock as well as enforced by law in many countries. We present a new integer programming formulation that links a hyperflow to model vehicle compositions and their coupling decisions to a set of path variables that take care of the resource consumption of the individual vehicles. To solve the model we developed different column generation algorithms which are compared to each other as well as to the MILP flow formulation of [2] on a test set of real world instances.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization

Keywords and phrases Railway Rolling Stock Optimization, Integer Programming, Column Generation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2023.13

1 Introduction

The fundamental task of every passenger railway operator is to offer an attractive railway timetable to the passengers while operating it as cost efficiently as possible. The available rolling stock has to be assigned to trips so that all trips are operated, operational requirements are satisfied, and the operating costs are minimum. This so-called Rolling Stock Rotation Problem (RSRP) is well studied in the literature, for example in [6] or [4]; we refer to [12] for a detailed overview. An important aspect in optimizing railway rolling stock rotations is the scheduling of maintenance services. Indeed, regular maintenance is important for the safety and reliability of the rolling stock as well as enforced by law in many countries. However, each maintenance service causes additional costs not just for the service itself but also for deadhead trips to and from the maintenance location, and the opportunity costs arising from the unavailability of the vehicle to operate trips for the duration of the service. Therefore, integrating maintenance planning into rolling stock rotation planning is of central importance for finding efficient solutions. This holds particularly for railway companies that operate long-distance routes, where a vehicle typically does not end in the same depot after each day of operation. In the railway literature it is often the case that the considered models

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).
Editors: Daniele Frigioni and Philine Schiewe; Article No. 13; pp. 13:1–13:15
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and solution approaches are highly tailored to the specific requirements, operational rules, and setting of the respective railway operator. The way in which maintenance services are handled or not is no exception to that.

In [4] an arc-based and a path-based model to optimize rotations for instances modeling a cyclic one-day vehicle schedule of a regional railway operator were presented. Compositions of different train types were considered in the sense that the number of vehicle types in a composition is taken into account, but without explicit handling of couplings. Maintenance is considered in the path-based model in the sense that for each maintenance constraint and each vehicle type a certain share of the paths must contain a maintenance service. Both models where solved by an LP-based heuristic.

Rolling stock rotations for a single vehicle type with a fixed composition for a cyclic one-day planning horizon are studied in [7]. Turns and maintenance services are determined by a MILP Model that uses a resource flow to track the resource consumption of each vehicle; it is solved by a commercial MILP-Solver.

A path-based mixed integer program is introduced in [10] in order to find rolling stock rotations for the S-tog trains in Copenhagen. Deadhead trips are not considered and an explicit predecessor-successor relation is considered for turns between trips. Although coupling is not modeled explicitly, the order of vehicles in a composition is, and composition changes are possible. Maintenance services are considered to be carried out after the vehicle arrives at a depot, and unit specific distance limits are included as a maximum distance threshold. The presented model is solved by a branch and price algorithm combining column generation and branch and bound.

[13] consider fixed maintenance services which are already integrated into the timetable and which a fraction of the vehicles have to visit. Three MIP formulations that are based on the flow model of [6] are introduced to optimize short re-scheduling situations for scenarios of Nederlandse Spoorwegen. The models very explicitly take into account multiple vehicle types and model coupling and decoupling as well as the position of each vehicle in a composition. A MIP solver is used to solve the models.

A two-stage MILP approach is presented by [14] to optimize a cyclic two day planning horizon of the Chinese high speed railway system. In the first stage an adaptation of [6] is used to compute optimized rotations for vehicle types, followed by a second MILP-stage to assign maintenance-feasible trip sequences to individual vehicles.

In this paper we present a novel integer linear formulation to model the Rolling Stock Rotation Problem with maintenance constraints as well as approaches to tackle the resulting model. Though being based on the work of [2] where a mixed integer linear program, based on a graph-based hypergraph, was developed to optimize rolling stock rotations, the model presented here uses path-based variables to take care of the resource consumption of individual vehicles instead of using an arc based resource flow. In contrast to [10] positions of vehicle types in operated vehicle compositions and their impact on turnings between the trips are considered. As the model contains exponentially many variables if all paths variables were added explicitly, different column generation algorithms are presented to solve a model with a suitable selection of path and hyperarc variables.

The paper is structured as follows. Section 2 presents a description of the hypergraph that is used to model the RSRP and a novel integer linear programming formulation to solve it. Section 3 describes several column generation algorithms that we use to tackle this formulation. In Section 4 we show the results of our computational study, that gives a comparison of the column generation algorithms and the approach of [2] on a test set of real world instances. Finally, a conclusion and outlook is given in Section 5.

2 Solving the RSRP with Maintenance Paths

We tackle the Rolling Stock Rotation Problem with maintenance constraints in a very similar way as [2] or [8], but with a different, namely, a path-based handling of the resource consumption of the individual vehicles. Among other ideas, [2] employed a coarse-to-fine approach where a part of the problem is solved on a less detailed coarse hypergraph layer, and the coarse solution is used to find a solution to the original problem on the fine hypergraph layer more efficiently. In the hypergraph model of [2] and [8], a binary hyperflow is used to compute the vehicles movement and shunting decisions. An additional arc flow linked to the hyperarcs is used to track the resource consumption for each individual vehicle. However, the linear relaxation of this model allows that fractions of vehicles are maintained such that the model systematically underestimates the number of maintenance services. This in turn means that the lower bound provided by the linear relaxation is not very tight and, as we have observed, can even schedule more maintenance services than the integer optimum. To overcome this drawback, we present a path-based model of the Rolling Stock Rotation Problem which provides a lower bound that is at least as tight as or tighter than the lower bound provided by the flow-based model of [2]. To solve the model, we developed multiple column generation algorithms which compute feasible paths, with respect to maintenance rules, in a coarsened graph. The approaches were tested on real world instances for an intercity railway network.

The ILP model used in this paper can be described as follows. Let T be the set of trips in the timetable. We consider the RSRP as given by a graph-based hypergraph $G := (V, A, H)$ where V is a set of nodes, A a set of standard arcs, and H a set of hyperarcs. V contains nodes for arrival or departure events of vehicles that operate trips in certain compositions of vehicles, or events where vehicles become available or are required at begin or end of the planning horizon, respectively. Let $M \subset V$ be a set of service events where maintenance services can be performed. The arc set A contains a standard arc (v, w) if a vehicle of the respective type can transfer in an operationally feasible way from v to w. The hypergraph H also contains hyperarcs $h \in H$, which are node disjoint subsets of A. If two arcs $(a, b), (v, w) \in h \subset A$ belong to a hyperarc h, this models the coupled transfer of two vehicles from a to b and from v to w, respectively. For more details concerning the construction of such a graph-based hypergraph we refer to [2]. Different from [2], but according to [8], we consider an acyclic setting with a time horizon, which leads to the consideration of start and end conditions for the rolling stock. Therefore let $S, E \subset T$ define sets of dummy trips modeling these conditions. For a start condition dummy trip $s \in S$, the trip's arrival node is the location of the respective vehicle at the beginning of the planning horizon. For an end condition dummy trip $e \in E$ the trip's departure node is a location where a vehicle of that type can be parked at the end of the planning horizon. Moreover, there are cost and resource functions $c : H \to \mathbb{Q}$ and $r : H \to \mathbb{Q}$ that give the cost to operate and the resource consumption with respect to maintenance services of a hyperarc h, respectively. The resource consumption of trip $s \in S$ is the initial level of resource consumption of the vehicle, while trips $e \in E$ require an extra resource buffer amount that must be kept available. Finally, $H_M \subset H$ defines the set of hyperarcs that include maintenance services. The RSRP is the task of finding a cost minimal hyperflow in G such that each sub-path of standard arcs between two hyperarcs of H_M is maintenance-feasible, i.e., taht the sum of resource consumptions along this path is below a certain threshold $R \in \mathbb{Q}$.

Figure 1 illustrates the hypergraph construction. It shows a snippet of a hypergraph that models four trips. Each node refers to an arrival or departure event of a single vehicle. Departure events are on the left hand side of the smallest surrounding box while arrival events

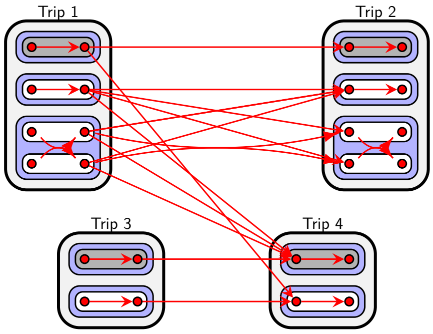


Figure 1 An Exmaple Hypergraph Modeling Four Trips.

are the nodes on the right. The trips 3 and 4 can be operated with a single vehicle composition either in orientation *tick* (1st class is in front) or *tock* (2nd class is in front), the former is shown as a single red arc surrounded by a white box while the latter is shown as a single red arc surrounded by a gray box. Compositions itself are grouped into a surrounding blue box. So there are two blue composition boxes for each of the two trips modeling two options to operate them. Trips 1 and 2 can additionally be operated by a two vehicle composition with orientation *tick* for both vehicles. Thus there are two white boxes surrounded by a blue box. The four nodes – two arrival and two departure events – are connected by a single hyperarc connecting the four nodes. All possible compositions to operate a trip are then surrounded by a white box headlined with the trip's name. The example shows the possible turnings of the vehicles between the four trips. If for example trip 1 is operated by a single vehicle composition with orientation *tock* (gray box), it has to be either succeeded by trip 2 operated by a single vehicle composition with orientation *tock* or trip 4 operated by a single vehicle composition with orientation *tick*. Therefore there is an orientation change for the turn between trip 1 and 4 while there is none between 1 and 2. A reason for that could be a different direction, in which a vehicle has to depart when it operates trip 2 or 4, or an additional deadhead trip for one of the two turns. Similarly, if trip 1 is operated by a single vehicle composition with orientation *tick* (white box), it has to be either succeeded by trip 2 operated by a single vehicle composition with orientation *tick*, or trip 4 operated by a single vehicle composition with orientation *tock*. Additionally, it can also be coupled to one of the two positions of the two-vehicle composition by which trip 2 can be operated. Finally, if trip 1 is operated by a two vehicle composition with orientation *tick* for both vehicles (white boxes), the vehicles can either procceed in two-vehicle composition of trip 2 using the hyperarc that connects the two arrival nodes of trip 1 with the two departure nodes of the two-vehicle composition of trip 2, or the composition can be uncoupled such that one vehicle is assigned to trip 2 and the other to trip 4. So there must be an orientation change for turns of vehicles from Trip 1 to Trip 3 while vehicles that turn from Trip 1 to Trip 2 maintain their orientation.

2.1 A Path-Based Integer Linear Programming Model to the RSRP

Here is an integer programming model of the RSRP. We denote by $H(t) \subset H$ the set of hyperarcs that operate trip $t \in T$, by $H(a) \subset H$ the set of hyperarcs that contain arc $a \in A$, and by $P(a)$ the set of maintenance feasible paths in (V, A) that contain arc $a \in A$. The model contains three different types of integer decision variables, namely, x_h for all $h \in H$, z_p for all $p \in P$, where P denotes the set of maintenance feasible paths in (V, A), and slack variables s_t for $t \in T$.

$$\min \sum_{h \in H} c_h x_h + \sum_{t \in T} M s_t \tag{RSRP$_{path}$}$$

$$s.t. \quad \sum_{h \in H(t)} x_h + s_t = 1 \qquad \forall t \in T, \tag{1}$$

$$\sum_{h \in H(a)} x_h - \sum_{p \in P(a)} z_p = 0 \qquad \forall\, a \in A, \tag{2}$$

$$\sum_{p \in P_m^+} z_p - \sum_{p \in P_m^-} z_p = 0 \qquad \forall\, m \in M, \tag{3}$$

$$s_t \in \{0, 1\} \qquad \forall\, t \in T, \tag{4}$$

$$x_h \in \{0, 1\} \qquad \forall\, h \in H, \tag{5}$$

$$z_p \in \{0, 1\} \qquad \forall\, p \in P. \tag{6}$$

The objective function (RSRP$_{path}$) minimizes the costs of vehicle movements associated with the chosen hyperarcs and penalties resulting from uncovered trips. Constraints (1) stipulate that each trip is either operated by a suitable composition hyperarc or that slack costs are paid. In case of a dummy trip for start or end conditions the slack costs are zero. Constraints (2) make sure that each standard arc contained in a chosen hyperarc is covered by a feasible maintenance path, and that flow conservation holds. The equalities (3) handle the conservation of paths entering and leaving a maintenance service location; these constraints are only considered in some of our algorithms, namely, those in which generated paths are split into subpaths at each visited maintenance service location. Finally, constraints (4),(5), and (6) define the variable domains.

The model potentially contains an exponentially large number of path variables. We therefore developed a number of column generation procedures to generate promising maintenance paths in order to solve the linear programming relaxation of this formulation.

3 Column Generation Approaches to the Path-Based ILP Formulation

To tackle the RSRP$_{path}$-formulation we run a column generation approach with different schemes to dynamically generate promising maintenance-feasible paths. Column generation is a technique best suited to solve MILP formulations with a very large set of variables compared to the number of constraints. It is based on the observation that there are very few basic variables in an optimal solution and that most others are zero. In a nutshell a so called restricted master problem – usually the original problem restricted to a subset of variables – is solved to obtain a primal and a dual solution vector $\bar{x}$ and π, respectively. Based on the dual information a pricing problem is solved to find variables with negative reduced cost. If there are no such variables the actual primal incumbent can not be improved anymore and is thus optimal. Otherwise variables with negative reduced cost are added to the restricted master problem and the next iteration begins. For deeper insights on the topic of column generation we refer to [5]. In our application the restricted master problem RSRP$_{res}$ is the RSRP$_{path}$-formulation restricted to the variables s_t for all $t \in T$ and x_h for all $h \in H_t$, the constraints (1) and (2) where already variables are present, and the constraints (3) for nodes $s \in S \cup E$. All other constraints are added at the time when one of the associated variables is added.

Listing 1 Algorithm 1: Column Generation Algorithm.

```
Input : Hypergraph G = (V, A, H), cost function c, resource function r,
maximum tolerance for optimality gap ε, number of vehicles k
Output: Generated paths P' and hyperarcs H' such that RSRP_res has
optimality gap of at most ε
Initialize: H' ← H_T, P' ← ∅, L ← 0
do
π ← dualSolve(RSRP_res(H', P'))
c̄ ← calculateReducedCostFunction(c, π)
P* ← calculateShortestMaintenancePathsTemplate((V, A), c, r)
if c̄(p) ≥ 0 ∀p ∈ P* then
  break
end
P' ← P' ∪ P*
H' ← H' ∪ ⋃_{a∈p∈P} H(a)
x* ← objectiveValue(π)
L ← max(x* + k min{c̄(p)|p ∈ P}, L)
while (x* − L)/x* > ε
return H', P'
```

In the pricing problem we have to check for promising variables x_h for $H \setminus H_t$ and z_p for all maintenance feasible paths $p \in P$ with negative reduced cost. In the latter case this can be done by solving the minimization problem

$$c_P^* := \min\Big\{\sum_{a\in p} c(a) + \sum_{a\in p} \pi_a \,|\, p \in P\Big\},$$

which is a resource constrained shortest path problem in $D = (V, A)$ with cost function $\hat{c} : A \to \mathbb{Q}, \hat{c}(a) := c(a) + \pi_a$ and resource function r. As it is possibly the case that the restricted master problem RSRP_{res} does not yet cover some arcs $a \in A$ by at least one hyperarc, we compensate for that by using the cost function

$$\bar{c} : A \to \mathbb{Q}, \bar{c}(a) := \begin{cases} c(a) + \pi_a & \forall a \in A : \exists h \in H_{\text{RSRP}_{res}} : a \in h, \\ c(a) & \text{else,} \end{cases}$$

where $H_{\text{RSRP}_{res}}$ denotes the set of hyperarcs present in RSRP_{res}. Solving the optimization problem

$$\bar{c}_P^* := \min\Big\{\sum_{a\in p} \bar{c}(a) \,|\, p \in P\Big\}$$

gives a set of promising hyperarc and path variables to add to RSRP_{res}in case of $\bar{c}_P^* < 0$, or proves that the column generation process can be stopped. The pseudo code for this algorithm is given in Algorithm 1.

Column generation often suffers from so-called tailing off: The closer the objective value of the incumbent approaches the optimal objective value, the smaller becomes the improvement of the objective function in each iteration. We therefore apply an additional stopping criterion in terms of a progress threshold. It applies when $\frac{c(\bar{x}) - k\bar{c}_P^*}{c(\bar{x})} \leq \varepsilon$, where ε is a given threshold and $k := |S|$ is the number of vehicles (of the respective type).

3.1 Coarsening Projections for the RSRP Hypergraph

Our algorithms are based on the previously mentioned hypergraph coarsening scheme developed by [11]. In the node set V of our original hypergraph G, a node $v \in V$ represents an arrival or departure event $e \in \{a, d\}$ of a vehicle of some type r operating a trip $t \in T$ in a chosen composition q at position i with orientation o. This node can be represented by a tuple $v := (e, t, r, q, i, o)$. The first coarsening of the hypergraph G is defined by the mapping

$$[\cdot] : V \to [V], [(e, t, r, q, i, o)] := (e, t, r, q)$$

which omits the position and the orientation of the node. We accordingly coarsen the arc and hyperarc sets to

$$[A] := \{([v], [w]) \in [V]^2 \,|\, \exists\, (v, w) \in A\} \quad \text{and} \quad [H] := \{ \bigcup_{(v,w)\in h} ([v], [w]) \,|\, \exists\, h \in H\}.$$

These three sets define a coarsened hypergraph $[G] := ([V], [A], [H])$, which we call the *configuration layer*. Similarly, we define a third layer called the *vehicle layer* by the mapping

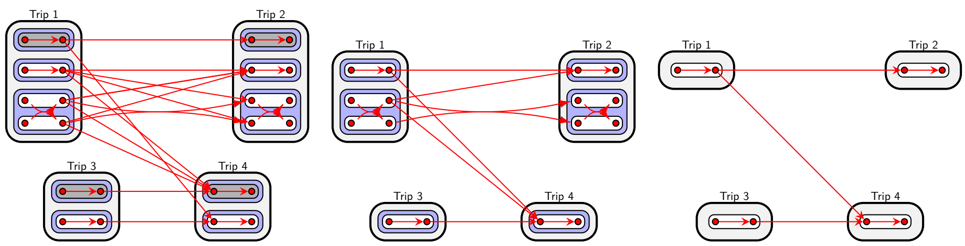
$$[[\cdot]] : V \to [[V]], [[(e, t, r, q, i, o)]] := (e, t, r),$$

which additionally omits the composition. The sets of arcs and hyperarcs of the *vehicle layer* $[[G]] := ([[V]], [[A]], [[H]])$ are defined as

$$[[A]] := \{([[v]], [[w]]) \in [[V]]^2 \,|\, \exists\, (v, w) \in A\} \quad \text{and} \quad [[H]] := \{ \bigcup_{(v,w)\in h} ([[v]], [[w]]) \,|\, \exists\, h \in H\}.$$

The costs of a hyperarc belonging to one of the coarse layers are conservatively defined as $c : [H] \to \mathbb{Q}, c([h]) := \min\{c(h') \,|\, h' \in H : [h'] = [h]\}$ and $c : [[H]] \to \mathbb{Q}, c([[h]]) := \min\{c(h') \,|\, h' \in H : [[h']] = [[h]]\}$, respectively.

The idea behind these graph contractions is that the coarsened graph becomes much smaller, but hopefully looses only little information, such that algorithms will run faster on the coarse graph, but still generate fesaible solutions and, in particular, maintenance-feasible paths. We remark that the coarsening projections of arc, hyperarcs, and path always result in underestimations of their respective costs, i.e., $[c]([p]) < c(p)$ always holds.



(a) Fine layer H. **(b)** Configuration layer $[H]$. **(c)** Vehicle layer $[[H]]$.

Figure 2 An Example for the Layers built by $[\cdot]$ and $[[\cdot]]$ for the Hypergraph of Figure 1.

3.2 Generating Maintenance Feasible Paths Using Coarsened Hypergraphs

The most crucial part of every column generation algorithm is to generate the best suited new variables as fast as possible. A straight forward idea to come up with promising maintenance paths is to solve the induced resource constrained shortest path problem (SPPRC) which is a well studied problem, see [9] for more details. To this purpose, we implemented a Label Setting Algorithm that first computes a topological ordering of the nodes in the graph, then traverses the graph in this order and stores labels at each node for all Pareto-optimal sub-path. The pseudo-code for a version that returns the best n paths is shown in Algorithm 3. Remark that it is easy to handle the initial resource consumption of vehicles as this only requires to set the resource consumption variables of the initial labels to their respective values. Note that it is possible (though not required in our application) to enforce in this way at least one maintenance service stop for each vehicle, which is a constraint that is hard to include into the flow formulation of [2]. Using Algorithm 3 with $n = 1$ as the shortest path routine in Line 5 of Algorithm 1 results in our first column generation algorithm, which adds exactly one path per iteration.

To take better advantage of the layered structure of our hypergraph, we implemented an additional resource constrained shortest path algorithm whose pseudo code is given in Algorithm 4. The idea is the following: In each iteration of the column generation algorithm, the path search iteratively computes for each vehicle $i \in \{1, ..., k\}$ a coarse maintenance-feasible path q_i with minimum coarse reduced cost in the configuration layer by running Algorithm 3 on $[G] := ([V], [A], [H])$. After that, a fine maintenance feasible path p_i is again computed by Algorithm 3 on the subgraph (V_{q_i}, A_{q_i}) induced by q_i. The nodes V_{p_i} and all adjacent arcs of A are then removed from G before the next path for vehicle $i + 1$ is computed. If at least one maintenance feasible path p_i with $\bar{c}(p_i) < 0$ was generated, the set $P_i := \bigcup_{i=1}^{k} \{p_i\}$ is added to the variables of the RSRP$_{res}$. As it could be the case that there is no fine maintenance feasible path p_i in the subgraph induced by q_i, or all feasible ones are already added, we iterate through a set of shortest coarse paths until we find a feasible fine path. In our computational experiments this happens rarely. Both algorithms were implemented and evaluated in the master thesis [3].

Table 4 of the Appendix shows computational results for these two algorithms and shows that the algorithms are able to compute significantly better lower bounds for specific instances, but for a substantial price in terms of run time, and with the drawback that generated paths are often not able to cover all trips in an integer way. This motivated the development of a procedure that aims at a (more) simultaneous generation of paths.

3.3 Assignment Based Resource Constrained Path Generation Algorithm

The main algorithmic contribution of this paper is the Assignment Based Resource Constrained Path Generation Algorithm shown in Algorithm 2. It is motivated by the observation that the paths that are generated in later iterations, even if they have negative reduced costs, often lack complementary paths that are needed to cover all trips. The general idea behind the algorithm is to avoid this situation by simultaneously computing paths for the entire set of vehicles. This is done by solving an assignment problem that assigns a successor trip to each trip in the super-coarse layer $[[G]]$. The ensuing predecessor-successor relations result in implicit paths in G. Due to the integrality of the Assignment Problem they often produce an integral solution of the RSRP$_{path}$ – if all computed paths are maintenance feasible. In order to improve the lower bound or to terminate this method is combined with a single resource constrained shortest path computation.

The algorithm works as follows. At first a single iteration of Algorithm 4 is done to compute a single coarse resource constrained shortest path q in the configuration layer $[G]$. This path defines a subgraph $(V(q), A(q)) \subseteq G$, where $V(q)$ and $A(q)$ denote the sets of nodes and arcs that can be projected by $[\cdot]$ on nodes or arcs of q. In this subgraph a shortest maintenance feasible path p is determined by Algorithm 3 and added to P'. If no such

Listing 2 Algorithm 2: Assignment Based Path Generation.

```
Input : Hypergraph (V, A, H), cost function c, gap
tolerance ϵ, number of vehicles k, coarseining projections [·], [[·]]
Output: Sets H′ ⊂ H and paths P′ ⊂ P with min{c̄(p)|p ∈ P′} < 0 or P′ = ∅
Initialize: H′ ← ∅, P′ ← ∅, L ← 0
π ← dualSolve(RSRP_res(H′, P′))
H′, P′ ← calculateCoarse2FineShortestPathSet(k = 1, [n] = 64, n=1)
if P′ = ∅ then
  break
end
[[c̄]] ← calculateSuperCoarseReducedCostFunction(c, π)
[c̄] ← calculateCoarseCostFunction(c, π)
c̄ ← calculateFineCostFunction(c, π)
[[A]] ⊃ A′ ← solveAssignment( ([[V]], [[A]]), [[c̄]])
[[P]] ← computeMaintenanceFeasiblePathDecompostion(A′)
for [[p]] ∈ [[P]] do
  [G]_[[p]] ← ([V([[p]])], [A([[p]])])
  q ← calculateCoarseShortestPath([G]_[[p]], [c], [r], n = 1)
  p ← calculateFineShortestPath((A(q), V(q)), c̄, r, n = 1)
  P′ ← P′ ∪ {p}
  H′ ← H′ ∪ H(p)
end
return H′, P′
```

path exists or $\bar{c}(p) > 0$, the path generation is stopped. Otherwise we set up the following assignment problem for the vehicle layer $[[G]]$ and reduced cost function $[[\bar{c}]]$, adding arcs (v, v) to $[[A]]$ $\forall v \in [[S]] \cup [[E]]$ with $[[\bar{c}]](v, v) := 0$.

$$\min \sum_{a \in [[A]]} [[\bar{c}]](y_a) \tag{AP}$$

$$s.t. \quad \sum_{a \in [[A]]_v^+} y_a = 1 \qquad v \in [[V(t)]],\ \forall t \in T, \tag{7}$$

$$\sum_{a \in [[A]]_v^-} y_a = 1 \qquad v \in [[V(t)]],\ \forall t \in T, \tag{8}$$

$$y_a \in [0, 1] \qquad a \in [[A]]. \tag{9}$$

The two sets of constraints (8) and (7) AP assign to each node of the vehicle layer a predecessor and a successor. The objective function ensures that this is done in a cost minimal way according to the super coarse reduced cost $[[\bar{c}]]$. Remark that each trip $t \in T$ has exactly two nodes in $[[V]]$, an arrival and a departure node. The problem is solved with an implementation of the Primal Hungarian Method of [1]. Because of the acyclic structure of the graphs, the solution translates into a set of paths $[[P]]$ (and loops for unused start or end vehicles) in the vehicle layer. Each path $[[p]]$ defines disjoint subgraphs $([V([[p]])], [A([[p]])])$ of the configuration layer $[G]$ that can be projected onto $[[p]]$. For each of these subgraphs we compute a shortest coarse maintenance feasible path q with respect to $[\bar{c}]$ with Algorithm 3, and from that a shortest maintenance feasible path p with respect to $\bar{c}$, which are added to P'. Denote the set of generated hyperarcs by $H' = \{h \in H \mid \exists a \in p : a \in h\}$. The sets P' and H' are returned as promising variables for the next iteration of the column generation round.

Table 1 Characteristics of the Test Instances.

Instance	$\lvert T\rvert$	$\lvert M\rvert$	$\lvert[V]\rvert$	$\lvert V\rvert$	$\lvert[H]\rvert$	$\lvert H\rvert$
Instance 1-3	215	5	259	518	40362	159286
Instance 4-6	267	4	315	630	66227	264054
Instance 7-8	274	4	274	548	70702	281748
Instance 9-11	276	4	276	562	71123	283406
Instance 12-14	276	4	276	562	71362	284402
Instance 15-17	284	4	284	568	75602	301362
Instance 18-20	62	20	69	138	3374	13414

3.4 Improving the IP by Using Subpaths

In all our approaches to solve the RSRP, we generate paths to solve the root LP. This can cause problems for the solution of the IP, as it might be hard to find subsets of path that jointly cover all trips. To overcome problem this we implemented for all of our algorithms a variant that splits each s-e-path at each maintenance service location into a set of subpaths. These subpaths are added to the master problem, coupled together by a path conservation constraint (3) for each maintenance service and each split, repsectively. Note that we can ignore the dual variables of the constraints (3) becasue we are still computing maintenance feasible paths with minimum reduced cost from a start trip to an end trip. The reason for that is that it is not possible to end a path at a maintenance service stop. Thus there must be exactly one subpath that enters and exactly one that leaves the maintenance service stop such that the associated dual variables cancel in an s-e-path. The construction increases flexibility in future iterations as solutions can be combined from subpaths, and are not limited to the set of generated s-e-paths. Note also that it is not possible to construct a path that is not maintenance-feasible. We refer to the variant of Algorithm 2 using maintenance-feasible subpaths as Algorithm 2+subpath in Section 4.

4 Computational Results

We evaluate all our algorithms on real world test set of long distance rolling stock rotation problems. All instances model an acyclic planning horizon of one week. Turnings including deadheads and additional turnaround trips are possible between each pair of trips as long as time and spacial constraints are met. Additional characteristics of the instances and the resulting numbers of coarse and fine nodes and hyperarcs are given in Table 1. In all of our instances we consider an initial resource consumption level of 0. Despite this idealized setting, almost all (optimal) solutions of the considered instances require each vehicle to have at least one maintenance service during the one week planning horizon.

All of our column generation procedures were implemented in the software tool ROTOR which is a railway rolling stock optimizer developed at ZIB and described in [11]. We ran our column generation routines with a run time limit of 2 hours to solve the linear programming relaxation. Afterwards the resulting IP formulation RSRP$_{res}$ is solved without any additional generation of paths. In spite of ROTOR's tailor-made branching scheme, we use CPLEX with a run time limit of 4 hours to have a more accessible comparison. All computations were performed on Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz and 64 GB of RAM. All restricted master problems that arise during column generation as well as the integer program that results from the column generation were solved using CPLEX 12.8.0.0 with an IP tolerance of 0.01 and a maximum of four threads in parallel.

Table 2 Computational Results for the linear relaxation of the final RSRP_{res}.

Instance	Algorithm 2		Algorithm 2+subpath		Flow	
	CPU(s)	Bound	CPU(s)	Bound	CPU(s)	Bound
Instance 1	7220	0.970	7231	0.971	3	**1**
Instance 2	4283	1.000	3360	**1**	3	0.998
Instance 3	150	0.998	87	0.997	3	**1**
Instance 4	7271	0.858	7208	0.858	25	**1**
Instance 5	7251	0.979	4733	**1**	24	0.980
Instance 6	204	0.997	214	0.997	24	**1**
Instance 7	7234	0.952	2693	**1**	5	0.968
Instance 8	498	0.998	220	0.998	5	**1**
Instance 9	7220	0.855	7275	0.855	5	**1**
Instance 10	6314	0.997	2951	**1**	5	0.968
Instance 11	505	0.997	185	0.997	5	**1**
Instance 12	7278	0.856	7248	0.856	5	**1**
Instance 13	6279	**1**	2537	0.999	5	0.968
Instance 14	170	0.998	187	0.997	5	**1**
Instance 15	7222	0.845	7208	0.845	5	**1**
Instance 16	7209	0.997	3079	**1**	5	0.966
Instance 17	262	0.997	239	0.997	5	**1**
Instance 18	378	**1**	125	0.996	1	0.971
Instance 19	275	0.996	122	1.000	1	**1**
Instance 20	118	0.994	80	0.994	1	**1**

In Table 2 we compare the solution process for the linear relaxation of RSRP_{path} for three different algorithms: Algorithm 2 adding s-e-paths only, Algorithm 2+subpaths adding subpaths, and the Flow model of ROTOR. For each of the three algorithms Table 2 contains two columns. The columns headlined *CPU(s)* show the computation time of the column generation procedure of the respective algorithm in CPU seconds. The columns *Bound* show the best lower bounds of the algorithms relative to the best known bound that was computed by any of the three algorithms (marked by an integer 1 in bold font instead of a float 1.000). The comparison shows that although the path formulation gives in theory a better bound, it was only able to compute the best bound in 7 of the 20 cases, which is due to the run time limit. It becomes also clear that the better bound requires a lot of run time. Comparing the two versions of Algorithm 2 shows that the version where subpaths are added significantly outperforms the other version in sense of run time and bound quality; a results that is of course again related to the run time.

In Table 3 the characteristics of the solution process and the solutions of the final RSRP_{res} formulations is shown. For each of the three algorithms, Table 2 contains three blocks of columns headlined *CPU(s)*, *Cost*, and *Gap*. The first column marks the computation time in seconds that was required by the respective algorithm to solve the underlying IP formulation up to an LP-IP gap of 1% or to reach the run time limit. The *Cost* column contains relative costs compared to the minimum cost that was found by any of the three algorithms. The last column gives the LP-IP gap of the solutions found by the three algorithms compared to the best lower bound by any of the algorithms. Comparing the solution quality of the integer solutions shows that in 75% of the instances one of the path formulations finds an integer solution with a lower objective function value than the solution ROTOR computes. The cost and Gap column for Algorithm 2 shows a significant outlier for Instance 1. This is due to the fact that the computed solutions were not able to cover all trips in the timetable and thus have to use slack variables which have a huge impact on the objective function value. A direct comparison of the two variants of Algorithm 2 reveals that although the version without adding subpaths is able to best solve 10 compared to 7 instances, it still gets outperformed by the variant that add subpaths as the latter one computes superior solutions it the sense of lower average objective function values.

Table 3 Computational Results for Solution Process of the Final IP of RSRP_{res}.

	Algorithm 2			Algorithm 2+subpath			Flow		
Instance	CPU(s)	Cost	Gap	CPU(s)	Cost	Gap	CPU(s)	Cost	Gap
Instance 1	14420	53,816	6297,00	14431	1,193	41,79	14404	**1**	18,87
Instance 2	11483	1,004	2,32	8474	**1**	1,89	14404	1,011	2,97
Instance 3	151	**1**	0,49	88	1,000	0,52	13	1,003	0,81
Instance 4	14472	1,161	52,03	14409	**1**	30,90	14422	1,019	33,35
Instance 5	7371	1,007	1,62	4765	**1**	0,94	14408	1,010	1,97
Instance 6	205	**1**	0,57	215	1,003	0,89	12	1,001	0,65
Instance 7	7280	1,011	1,70	2696	**1**	0,62	14420	1,006	1,27
Instance 8	501	**1**	0,58	220	1,001	0,67	51	1,004	1,00
Instance 9	14420	1,055	40,06	7296	**1**	32,74	14407	1,050	39,40
Instance 10	6325	**1**	0,47	2954	**1**	0,47	14411	1,003	0,80
Instance 11	509	**1**	0,57	185	1,001	0,68	25	1,003	0,83
Instance 12	7287	1,000	35,62	7319	1,000	35,62	14410	**1**	35,61
Instance 13	6304	**1**	0,81	2544	1,004	1,19	14410	1,003	1,14
Instance 14	171	**1**	0,54	187	1,001	0,65	43	1,001	0,66
Instance 15	7229	1,000	36,87	7216	1,001	36,99	14409	**1**	36,87
Instance 16	7223	**1**	0,77	3082	**1**	0,77	14420	1,000	0,78
Instance 17	263	**1**	0,55	240	1,001	0,66	44	1,002	0,72
Instance 18	382	1,001	1,41	162	1,133	14,79	14402	**1**	1,28
Instance 19	277	**1**	0,75	132	1,010	1,79	4794	1,008	1,53
Instance 20	119	1,200	20,93	82	1,002	1,03	1	**1**	0,80

5 Conclusion and Outlook

In this paper we presented a novel path based ILP-formulation to the Rolling Stock Rotation Problem as well as sophisticated column generation algorithms to tackle the Problem. Although the presented algorithms were designed with the focus of generating tight lower bounds for the Rolling Stock Rotation Problem with maintenance constrains, it turns out that it was even possible to compute high quality integer solutions for practically relevant instances, albeit for the price of longer running times as compared to the flow model of ROTOR. Moreover, we were able to solve the instances considered in this paper by solely generating paths, respectively subpaths, of the root relaxation, without any additional path generation later on in the branching tree. This favorable outcome might be a consequence of a good overall fit of the generared paths, which in turn is caused by extended degrees of freedom from the subpath construction and the generation of maintenance-feasible paths with a fleet focus in the assignment based generation approach. Additional research is needed to further improve the run time of the path generation.

References

1. M. L. Balinski and R. E. Gomory. A Primal Method for the Assignment and Transportation Problems. *Management Science*, 10(3):578–593, April 1964. `doi:10.1287/mnsc.10.3.578`.
2. Ralf Borndörfer, Markus Reuther, Thomas Schlechte, Kerstin Waas, and Steffen Weider. Integrated optimization of rolling stock rotations for intercity railways. *Transportation Science*, 50(3):863–877, 2016. `doi:10.1287/trsc.2015.0633`.
3. Julian Bushe. Rolling stock rotation optimization with maintenance paths. Master's thesis, Technische Universität Berlin, 2021.
4. Valentina Cacchiani, Alberto Caprara, and Paolo Toth. Solving a real-world train-unit assignment problem. *Mathematical Programming*, 124(1):207–231, July 2010. `doi:10.1007/s10107-010-0361-y`.
5. Jacques Desrosiers and Marco E Lübbecke. A primer in column generation. In *Column generation*, pages 1–32. Springer, 2005.

6 Pieter-Jan Fioole, Leo Kroon, Gábor Maróti, 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. doi:10.1016/j.ejor.2005.03.032.

7 Giovanni Luca Giacco, Andrea D'Ariano, and Dario Pacciarelli. Rolling stock rostering optimization under maintenance constraints. *Journal of Intelligent Transportation Systems*, 18(1):95–105, 2014. doi:10.1080/15472450.2013.801712.

8 Boris Grimm, Ralf Borndörfer, Markus Reuther, Stanley Schade, and Thomas Schlechte. A propagation approach to acyclic rolling stock rotation optimization. In *7th International Conference on railway operations modelling and Analysis (RailLille 2017)*, pages 688–698, 2017.

9 Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In *Column generation*, pages 33–65. Springer, 2005.

10 Richard M. Lusby, Jørgen Thorlund Haahr, Jesper Larsen, and David Pisinger. A branch-and-price algorithm for railway rolling stock rescheduling. *Transportation Research Part B: Methodological*, 99:228–250, 2017. doi:10.1016/j.trb.2017.03.003.

11 Markus Reuther. *Mathematical Optimization of Rolling Stock Rotations*. PhD thesis, Technical University Berlin, 2017. URL: https://depositonce.tu-berlin.de/handle/11303/6309.

12 Per Thorlacius, Jesper Larsen, and Marco Laumanns. An integrated rolling stock planning model for the Copenhagen suburban passenger railway. *Journal of Rail Transport Planning & Management*, 5(4):240–262, 2015. doi:10.1016/j.jrtpm.2015.11.001.

13 Joris C. Wagenaar, Leo G. Kroon, and Marie Schmidt. Maintenance appointments in railway rolling stock rescheduling. *Transportation Science*, 51(4):1138–1160, 2017. doi:10.1287/trsc.2016.0701.

14 Qingwei Zhong, Richard M. Lusby, Jesper Larsen, Yongxiang Zhang, and Qiyuan Peng. Rolling stock scheduling with maintenance requirements at the Chinese high-speed railway. *Transportation Research Part B: Methodological*, 126:24–44, 2019. doi:10.1016/j.trb.2019.05.013.

Listing 3 Algorithm 3: Calculate Shortest Path Labels with Resource Constraints.

```
Input : Graph G=(V, A), topological ordering of V, start and end node
sets S and E, cost function c, resource function r, integer n.
Output: Label of k minimum cost maintenance feasible s-e-path
//initialize empty pareto sets at each node
foreach v ∈ V do
  labels(v) ← ∅
end
bestEndLabelsList ← sortedListOfLength(n)
//create labels at initial departure nodes
foreach v ∈ V do
  label ← createStartLabel(v)
  labels(v).add(label)
end
//relax outgoing arcs of nodes in topological order
for i ∈ {1,...,n} do
  if v_i ∉ E then
    foreach a = (v_i, w) ∈ A+_{v_i} do
      foreach label ∈ labels(v_i ) do
        newLabel ← createLabel(label, a, c, S)
        if newLabel is feasible then
          if newLabel is not dominated of lables(w) then
            labels(w).add(newLabel)
            labels(w).discardLabelsDominatedBy(newLabel)
          end
        end
      end
    end
  end
  //if v_i is a terminal arrival node
  else
    //labels at each node ordered by ascending cost
    for candidateLabel in labels(v_i) do:
      if cost(candidateLabel) < cost(bestEndLabel) then
        bestEndLabelsList.insert( candidateLabel )
      end
    end
  end
end
return bestEndLabelsList
```

Listing 4 Algorithm 4: Calculate Coarse2Fine Shortest Path Set.

```
Input : Hypergraph (V, A, H), cost function c, gap
tolerance $\epsilon$, number of vehicles k, coarseining projection $[\cdot]$,
number of coarse paths [n], number of fine paths [n]
Output: Sets $H' \subset H$ and paths $P' \subset P$ with $\min\{\bar{c}(p) | p \in P'\} < 0$ or $P' = \emptyset$
Initialize: $H' \leftarrow H_T$, $P' \leftarrow \emptyset$, $L \leftarrow 0$
$\pi \leftarrow$ dualSolve(RSRP$_{res}$(H',P'))
$[\bar{c}] \leftarrow$ calculateCoarseReducedCostFunction($c, \pi$)
$\bar{c} \leftarrow$ calculateFineCostFunction($c, \pi$)
$Q \leftarrow \emptyset$ \\ Set of shortest coarse paths
$[G] \leftarrow ([V], [A])$
for $i = 1, ..., k$ do
  $Q_i \leftarrow$ calculateCoarseShortestPath($[G], [c], [r], [n]$)
  if $min\{[c](q) \,|\, q \in Q_1\} \geq 0$ then
    break
  end
  for $q_i \in Q_i$ do
    $P_{q_i} \leftarrow$ calculateFineShortestPath(($A(q_i), V(q_i)), \bar{c}, r, n$)
    if $|P_{q_i}| > 0$ then
      $P' = P' \cup P_{q_i}$
      $[G] \leftarrow$ deleteArcsAndNodesOfPath($[G], q_i$)
      break
    end
  end
end
if $\min\{\bar{c}(p) \,|\, p \in P'\} \geq 0$ then
  return $H', P'$
end
$H' \leftarrow H' \cup \bigcup_{p \in P'} \bigcup_{a \in A(p)} H(a)$
return $H', P'$
```

Table 4 Computational Results for Algorithm 3 and Algorithm 4.

	LP						IP				
	Relative Objective			CPU(s)			Relative Obj.		Gap(%)		
Id	F	A1	A2	F	A1	A2	A1	A2	F	A1	A2
1	0.992	0.992	0.992	6	18912	4705	54.0	21.43	0.8	98.2	95.3
3	0.865	0.912	0.912	13	71792	28441	48.3	38.90	8.8	98.1	97.7
4	0.994	0.997	0.997	34	70940	15739	43.5	1.000	0.3	97.7	0.3
6	0.758	0.885	0.885	34	66011	10923	21.6	6.822	11.4	95.9	87.0
7	0.759	0.991	0.992	21	99375	18538	27.9	0.972	3.6	97.6	0.9
9	0.995	0.999	0.999	21	48371	18179	27.1	0.996	0.4	96.3	0.1
11	0.786	0.999	0.999	10	60850	21799	19.4	0.950	5.0	95.1	0.1
12	0.996	0.999	0.999	17	49164	15475	33.5	0.995	0.5	97.0	0.1
14	0.748	0.970	0.970	25	154049	10850	35.6	10.19	3.0	97.3	90.5
15	0.991	0.992	0.992	1	297	191	197.2	1.196	0.7	99.5	17.1
18	0.959	0.991	0.991	1	520	493	147.9	1.216	0.9	99.3	18.5
20	0.996	0.999	0.999	18	65313	10362	33.0	0.998	0.2	97.0	0.1
17	0.996	0.998	0.999	23	76086	12305	48.3	0.996	0.4	97.9	0.1