Adaptive Large Neighborhood Search for Integrated Planning in Railroad Classification Yards

Moritz Ruf[a,*], Jean-François Cordeau[b]

[a]*Technische Universität Dresden, Institute of Railway Systems and Public Transport, 01062 Dresden, Germany*
[b]*HEC Montréal and CIRRELT, 3000 chemin de la Côte-Sainte-Catherine, Montréal, H3T 2A7 Canada*

Abstract

Railroad classification yards serve as central hubs in single wagonload freight transportation by disassembling inbound trains and classifying outbound ones. This enables railcars to switch trains, thereby reducing the number of point-to-point connections for low demand origin-destination pairs. The quality of the operations in classification yards has a large impact on the overall performance of the system. The planning process comprises the definition of railcar interchanges, the track allocation, the scheduling of service and safety operations, and the assignment of both locomotives and staff to operations. This tactical planning task is nowadays mainly done manually by experienced planners and most optimization models in the literature focus only on subproblems. In the hope of filling this gap, we therefore propose a formulation for the integrated planning problem in classification yards. Since the formulation turns out to be intractable for general-purpose solvers, we propose a tailored adaptive large neighborhood search heuristic that yields high-quality results for realistic instances. Problems with up to 20 inbound and outbound trains are solved on average in less than 20 minutes with an average optimality gap of 0.6% for the instances for which an optimal solution is known.

Keywords: Transportation; rail freight; single wagonload; adaptive large neighborhood search; railway optimization; railcar classification

1. Introduction

Rail freight offers a portfolio of several service options. Unit train services directly connect the origin and destination of a shipment. However, due to small amounts of freight and high costs, not all origin-destination pairs can be efficiently served by direct trains. In contrast, single wagonload (SWL) is a segment of rail freight that provides a service for "less than train load demand" (Marinov et al., 2012) on origin-destination pairs where a direct link is not practical (Fügenschuh et al., 2018). Its basic principle is that railcars are routed through the network in different trains. In spite of a decline during the last decades, the share of SWL in rail freight is still considerable in Central Europe (Marinov et al., 2012; European Commission, 2015).

The network of SWL in Europe is a modified hub-and-spoke system (Bruckmann, 2006; Mancera et al., 2015) and is typically composed by different hierarchical levels. The typical route of a railcar from origin to destination is as follows (see Fig. 1). First, the railcars are transferred from

*Corresponding author
Email addresses: moritz.ruf@tu-dresden.de (Moritz Ruf), jean-francois.cordeau@hec.ca (Jean-François Cordeau)

the loading point (e.g., an industrial railway) to the closest local hub. There, they are assembled with other railcars from different origins in a train heading to the closest classification yard. Classification yards are the largest hubs of the network. They are also referred to as *marshalling*, *hump* or *shunting yards* (Bohlin et al., 2018). Between the classification yards, long distance trains cover the long-haul legs of the railcars' routes. The railcars are either directly carried to the closest classification yard of the destination or via further classification yards. Finally, the railcars reach their delivery point via a local hub.

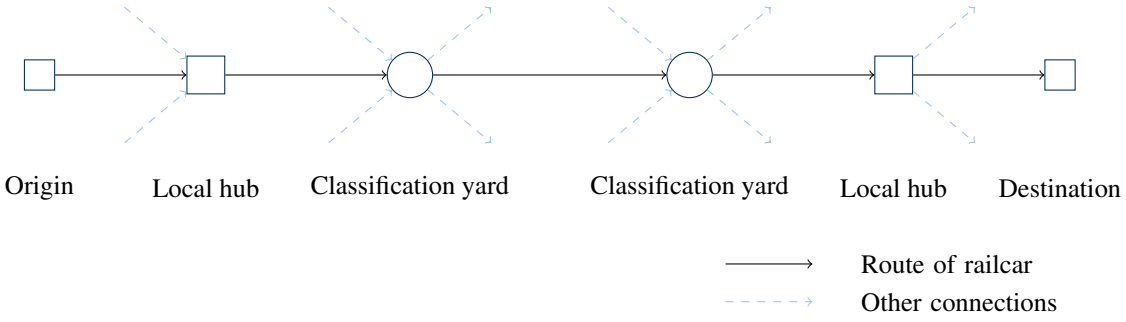


Figure 1: Trip of one railcar routed through a single wagonload transport network

As shown in Figure 1, railcars may switch from one train to another multiple times during their trip. This causes not only inevitable processing times related to technical requirements, but also waiting times for the connecting outbound trains. Obviously, the terminals can be a major source of reliability problems and a smooth operation of the processes in the terminals plays a crucial role for the overall efficiency of the SWL (Turnquist and Daskin, 1982).

A typical layout for a European classification yard is depicted in Figure 2. It has its pre-set working direction and consists of several yards: the *receiving*, the *classification*, and the *departure bowl*. Each bowl is equipped with a set of parallel tracks. An artificial hillock – the so-called *hump* – is located between the receiving and the classification bowl.

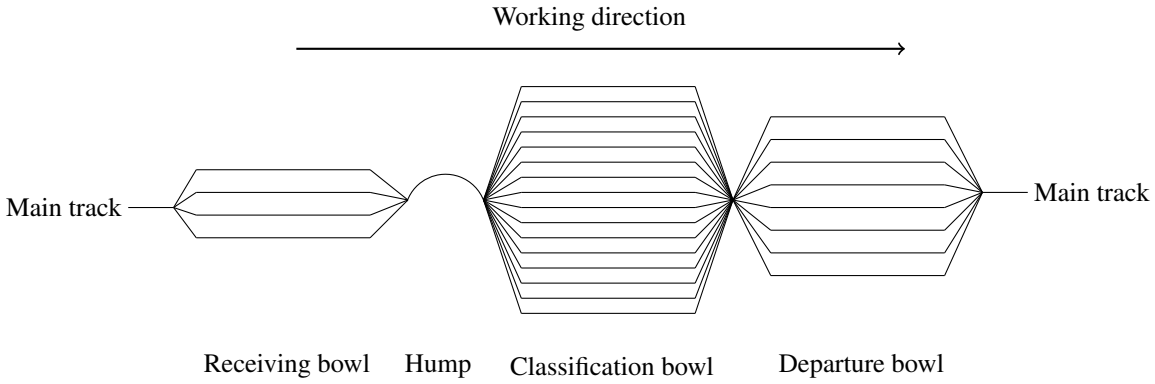


Figure 2: Schematic layout of a classification yard

The purpose of classification yards is to disassemble inbound trains, to sort the railcars according to their outbound directions, and to classify new outbound trains. Standard processes in European classification yards are as follows. First, inbound trains arrive from the main track in the receiving bowl. There, railcars that do not continue their path in the same outbound train are split. Next, the

uncoupled railcars are pushed over the hump by a shunting engine. The wagons run by gravity down the hump through a tree of switches to their assigned classification track. Retarders control their velocity. In the classification bowl, railcars with the same direction are coupled and their brake pipes are connected. Shunting locomotives pull the newly formed outbound train to the departing bowl. A wagon inspection and a brake check ensure the technical reliability of the outbound train. Finally, the outbound train departs from the classification yard (Boysen et al., 2012; Bohlin et al., 2018).

The processes require qualified staff and the railcars occupy infrastructure elements. Some processes need to be fulfilled by locomotives. Related to the local requirements, there exist many varieties of infrastructure layouts and processes. We refer to Belošević et al. (2015) and Petersen (1977) for in-depth overviews of classification yards.

Numerous planning and dispatching problems arise at the strategic, tactical and operational levels. The problems are related with the infrastructure assignment, the planning of railcar interchanges, the resource assignment, and the scheduling of the processes. Most of the related literature focuses on subproblems (see, e.g., Boysen et al., 2016, for a comprehensive review). This reduces the applicability for railway operators or yields suboptimal results.

Unlike most models in the operations research (OR) literature, we tackle an integrated tactical planning problem comprising all decision problems related with the creation of an operating plan. The problem consists in deciding which inbound blocks are assigned to which outbound train. Furthermore, all safety and service operations (e.g., uncoupling, brake check, etc.) need to be scheduled. Additionally, infrastructure elements, locomotives, and staff are allocated to the processes. The objective is to find a minimum number of resources needed for an operating plan for one day fulfilling all operational constraints. As the problem turns out to be difficult to solve with general-purpose solvers, we propose an adaptive large neighborhood search (ALNS) heuristic tailored for this problem.

The contributions of this paper are twofold:

- First, we provide a mixed integer linear program (MILP) including all four subproblems described by Boysen et al. (2016). Even though this is a highly relevant planning problem for operations, the existing literature is restricted to subproblems.

- Second, we propose an efficient metaheuristic that yields high-quality solutions for instances of realistic size. Problems with up to 20 inbound and outbound trains are solved on average in less than 20 minutes with an average optimality gap of 0.6% for the instances for which an optimal solution is known.

The remainder of the paper is structured as follows: Section 2 gives a review of operations in classification yards and summarizes the attempts made in the OR literature to tackle them. Section 3 provides the problem statement of the integrated tactical planning problem in classification yards. A formulation as a MILP is given in Section 4. We describe an ALNS heuristic tailored to this problem in Section 5. Computational results are reported in Section 6 and some concluding remarks end the paper in Section 7.

2. Literature review

Driven by an increasing competition in the sector, the efficient and rational planning of railroads has been the subject of research for many decades (Assad, 1981; Cordeau et al., 1998). To this day, the use of optimization in rail freight remains a very active field of research. For an overview on the latest developments, we refer to Boysen et al. (2012) and to Borndörfer et al. (2018).

2.1. The concept of blocking

A large share of the OR literature in rail freight is oriented towards the concept of *blocking*. This planning paradigm has been applied in for many decades, mainly in North America. The concept defines the (especially long-running) blocks between classification yards and the routes of the trains to which the blocks are assigned. The advantage is that railcars in blocks remain in a group over a long distance and, hence, the number of time-consuming and cost-intense reclassifications can be reduced.

In *classification yards*, blocks can be built and split. *Railcars* relate to orders that are routed through the network. *Blocks* are composed by railcars and start and end in classification yards. Railcars grouped in a block may not leave the block before its destination. Blocks left by a train in a classification yard are either split or transferred to another train. *Trains* – composed by multiple blocks – operate between classification yards and may stop at intermediate classification yards to set off or pick up blocks.

There are multiple optimization problems related to this concept. The *blocking policy* decides which blocks are built at each classification yard and which are their assigned railcars (Crainic et al., 1984; Bodin et al., 1980; Keaton, 1992; Ahuja et al., 2007; Crevier et al., 2012). The *train routing problem* decides on the routing and frequency of trains. The *train makeup problem* assigns blocks to trains on a tactical level. These two problems are typically solved as an integrated *train routing and makeup problem* (Assad, 1980; Haghani, 1987, 1989; Xiao et al., 2018). The *makeup policy* defines possible interchange between blocks in a specific classification yard. Additional related problems are the *freight car management* (Dejax and Crainic, 1987; Milenkovic and Bojovic, 2019), the *train dispatching* (Lamorgese and Mannino, 2015), the *track allocation* (Caimi et al., 2018), and the *locomotive assignment problem* (Cordeau et al., 2001; Piu and Speranza, 2014; Scheffler et al., 2020). The problem discussed in this paper relates to the train makeup problem, while considering many more important constraints related to the operation of a classification yard. It is a local problem at the tactical planning level.

Even though the concept of blocking is widespread in the literature, its application is questionable for European rail freight because of differences in operational, institutional and infrastructural constraints with respect to the North American market (see Posner III, 2008; Clausen and Voll, 2013; Mortimer and Islam, 2014, for a discussion). The different cost structure in Europe is reflected in the *single car routing problem* which replaces the *blocking policy problems* (Homfeld, 2012; Fügenschuh et al., 2015, 2018). Due to the density of the network, the necessity of timetables for freight trains, and the multitude of railway undertakings using the same infrastructure, *timetabling problems* and *track allocation problems* play a leading role in Europe (Lusby et al., 2011; Caimi et al., 2018; Cacchiani and Toth, 2012). Last, the concept of *block* is a rather artificial construct in Europe since the majority of trains in SWL end in classification yards without intermediate stops: classification yards, railcars and trains are the key elements for European optimization problems.

2.2. Operations research applications in classification yards

For reviews on queuing models, analytical models, and OR applications related to classification yards, we refer to existing surveys (Haghani, 1987; Cordeau et al., 1998; Boysen et al., 2012). The decision-making problems in yards can be classified into four subproblems (Boysen et al., 2016; He et al., 2003):

1. The *cut generation problem* assigns inbound trains to receiving tracks. As – especially in North America – the length of the train may exceed the length of the tracks, it may be necessary to decide how the inbound trains are divided (*cut*) to fit on the available tracks.

2. The *train makeup problem* assigns inbound railcars to outbound trains.
3. The *railcar classification problem* defines the humping sequence, decides on the allocation of classification tracks, finds a schedule and assigns shunting locomotives to the processes.
4. The *outbound track assignment problem* allocates outbound trains to departing tracks. If the train exceeds the track length, the outbound train is built on several departing tracks.

Based on these subproblems, we survey the related work. For problems in related areas, e.g., in passenger yards or in industrial railways, we refer to Kroon et al. (2008) and Lübbecke and Zimmermann (2005), respectively.

Both the *cut generation problem* and the *outbound track assignment* problem have received little attention in the literature. To the best of our knowledge, there is no study that solely focuses on one of the problems. They are part of integrated approaches (amongst others Boysen et al., 2016; Bohlin et al., 2012). The *train makeup* problem is methodologically similar to the multiple-choice knapsack problem with item fragmentation and is NP-hard (Boysen et al., 2016). Formulations together with both heuristic and exact solution techniques have been proposed by Li et al. (2014), Boysen et al. (2016), and Bektas et al. (2009).

The majority of the literature focuses on the railcar classification problem (Boysen et al., 2016), especially on the multi-stage sorting, where railcars may be pushed over the hump more than once, requiring a pullback from the classification bowl. A classification of sorting problems can be found in Di Stefano et al. (2007) and Hansmann and Zimmermann (2008), and we refer to Boysen et al. (2012) for a review along with an illustrative description of sorting strategies. In *strict single-stage sorting* problems, each railcar is humped only once. The most noteworthy contributions to this problem are Yagar et al. (1983) and Jaehn et al. (2015). The problem of *extended single-stage sorting*, where rehumpings are allowed in some cases, has been investigated in three papers by Kraft (Kraft, 2000, 2002a,b). *Multi-stage classification* is needed either when the order of cars in outbound trains is defined or if the number of classification tracks is insufficient. Sorting strategies define on which tracks the railcars are classified and in which humping and rehumping processes they are involved. Typical objectives are to minimize the number of classification tracks or the number of humping steps (Boysen et al., 2012; Bohlin et al., 2018). The algorithmic characteristics of multi-stage sorting along with different strategies are described in Gatto et al. (2009).

An early study can be found in Daganzo et al. (1983) where service time and space requirements for different sorting strategies are proposed. Noteworthy papers with models are Dahlhaus et al. (2000), Maue and Nunkesser (2009), and Jacob et al. (2011). The latter propose a binary encoding of the classification schedule using logical instead of real tracks. Some studies test their results in real-world applications (e.g., Márton et al., 2009). Bohlin et al. (2016) propose two integer linear program (ILP) formulations for a similar problem. Belošević and Ivić (2018) present an ILP that incorporates strategic multistage train classification aiming at optimizing the classification schedule and the number of sidings. They solve the problem with a variable neighborhood search (VNS). Some models consider the stochastic character of railway processes and focus on robustness of classification schedules in order to quickly recover from disturbances (Cicerone et al., 2009a,b; Büsing and Maue, 2010). The so-called *mixing problem* is investigated in three subsequent papers (Bohlin et al., 2011a,b, 2012). They focus on the allocation of classification tracks and the minimization of pulls of mixing-tracks that are necessary if the number of classification tracks is insufficient. The humping sequence along with start and end of the classification and the railcar interchanges are input to the problem.

Since the discussed decision problems in classification yards are highly intertwined, we now survey approaches that incorporate at least two of the aforementioned subproblems. He et al. (2000) describe an integrated approach for real-time dispatching in classification yards. The problem in-

volves the train makeup problem and the scheduling of both classification and assembly operations. They assume single-stage sorting. Some of the input parameters are considered fuzzy. They propose a genetic algorithm approach enriched by local search techniques. He et al. (2003) extend this model and introduce a decomposition method as a solution technique. To create the input for the mixing problem discussed before, an optimization problem which defines the humping schedule and the pull times is introduced in Bohlin et al. (2011a,b, 2012). Additionally, the problem allocates inbound and outbound trains to tracks. They thereby incorporate the cut generation, the railcar classification and the outbound track assignment problem in a MILP. The objective is to minimize the weighted time railcars spend in the classification bowl. Zhang et al. (2018) propose a MILP that incorporates the train makeup and the railcar classification problem providing a tailored tabu search. Preis et al. (2018) propose an integrated approach including the assignment of local resources to processes. Starting from given inbound and outbound trains, along with their railcar interchanges, they derive all required processes using extended single-stage sorting. They propose a MILP which schedules the processes and defines the infrastructure and resource utilization. The aim of the optimization problem is to minimize the weighted costs of necessary resources. Apart from the train makeup problem, they incorporate all other subproblems in one optimization problem. They propose a heuristic approach and a tailored exact solution technique based on Boolean satisfiability solvers. Raut et al. (2019) present a model coping with the train makeup, the railcar classification, and a part of the outbound track assignment problem. Additionally, they present heuristics that take care of the scheduling and assignment problem.

2.3. Summary of the literature review

Table 1 gives an overview of the relevant OR literature discussed above. The closest attention has been dedicated to the railcar classification problem. Some integrated models incorporate multiple subproblems. Most of them restrict themselves to track allocation.

To the best of our knowledge, there is no model so far that tackles all four subproblems defined by Boysen et al. (2016). The additional scheduling and assignment of processes is only discussed in Preis et al. (2018), which comes closest to our model. They incorporate the scheduling and assignment problem along with three out of four problems defined by Boysen et al. (2016). Unlike our setting, they assume that the railcar interchanges are an input to the problem, e.g., the train makeup problem is out of scope of their optimization. The integration of the train makeup problem widens the time windows of many requests, which yields an increase of the solution space that makes the problem harder to solve.

3. The Integrated Planning Problem in Classification Yards

3.1. Scope and objective of the problem

We consider a tactical planning problem in classification yards. It comprises all four subproblems defined by Boysen et al. (2016) and – at a finer level of detail – the assignment of local resources (infrastructure, staff, and locomotives) to processes.

Even though the problem is of high practical relevance, it has to the best of our knowledge not been tackled as an integrated model in the literature, yet. The relevant components defining the optimization problem are depicted in Figure 3. Its goal is to create a full operating plan comprising

- the railcar interchanges between inbound and outbound trains,
- the humping sequence,

	Bektas et al. (2009)	Bohlin et al. (2011a)	Bohlin et al. (2011b, 2012, 2016)	Boysen et al. (2016)	Daganzo et al. (1983)	Dahlhaus et al. (2000)	Gatto et al. (2009)	Hansmann and Zimmermann (2008)	He et al. (2000, 2003)	Jacob et al. (2011)	Kraft (2000)	Li et al. (2014)	Gestrelius et al. (2017)	Preis et al. (2018)	Raut et al. (2019)	Yagar et al. (1983)	Zhang et al. (2018)	*This contribution*
Problem																		
Cut generation		•											•	•				•
Train makeup				•					•			•			•		•	•
Railcar classification	•	•	•		•	•	•	•	•	•	•		•	•	•	•	•	•
Outbound track ass.		•							•					•	•			•
Scheduling and ass.														•				•
Objective																		
Dwell time	•	•							•			•			•			
Extra roll-ins			•		•		•			•			•				•	
Resources														•				•
Tracks						•		•					•	•				•
Other				•					•		•					•	•	•

Table 1: Overview of the literature on planning and dispatching problems in classification yards

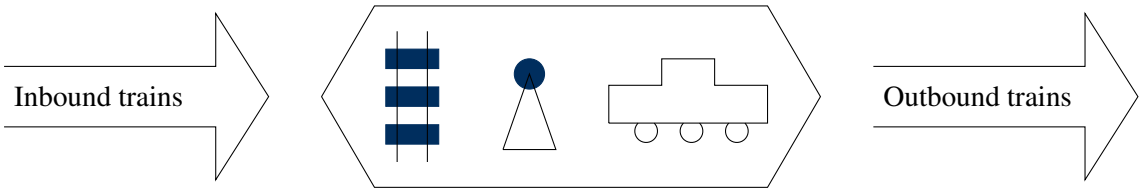


Figure 3: Input to the integrated tactical planning problem in classification yards

- the schedule of service and safety processes, and
- the assignment of resources to processes including track allocation.

The considered objective is to minimize the costs for used resources for an operating plan while ensuring that a maximum of required processes is executed.

3.2. Infrastructure

The model of the infrastructure can be adapted to any layout of classification yards. Each bowl (see Fig. 2) consists of a set of tracks having a specific length, given accessibilities from the main tracks, and other attributes, such as electrification. Each track has individual costs: e.g., an electrified track may be more expensive than a non-electrified one.

3.3. Resources

In addition to the infrastructure elements, we consider staff and locomotives. Each resource has individual qualifications or technical settings. It can be single- or multi-skilled. A maximum number

of resources that can be used in the operating plan is an input to the optimization problem. For each resource, individual transition times between two processes are given. As for tracks, each resource has its individual costs.

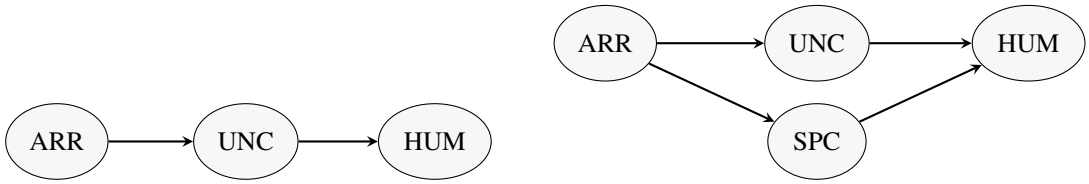
3.4. Trains

Each train consists of blocks. In this problem, we interpret a block as a group of railcars with a given destination entering the classification yard in the same inbound and leaving the yard in the same outbound train, e.g., the railcars of a block may not be split within the yard. The arrival time of inbound trains and the departure time of outbound trains is known and deterministic. Each outbound train has a given destination and a maximal capacity expressed in number of railcars.

3.5. Organizational assumptions

We define an *operational process* as an activity that needs to be accomplished by a given number of resources without interruption. It is defined by a type, a related train, a duration, a time window, and a required resource allocation (e.g., a shunter and a receiving track).

The processes must be fulfilled in a predefined order expressed by a process chain which is derived from the handling requirements of the train. The process chain can differ by train. Figure 4 depicts two example handling requirements for inbound trains. In Figure 4a, the handling of a basic inbound train without special requirements is shown. First, the train arrives (ARR) at the yard. Next, it needs to be uncoupled (UNC). Last, the train is humped (HUM). If the train contains a block that may not be humped (e.g., dangerous goods), the handling is completed by a special treatment (SPC, Fig. 4b). As this process does not interdepend with the UNC process, they can be executed in a parallel manner.



(a) Standard handling

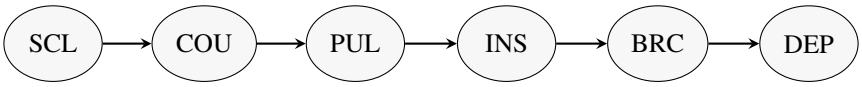
(b) Handling with special treatment

Figure 4: Example for two different handling requirements of inbound trains

Figure 5 shows two examples of handlings for outbound trains. For classical layouts of classification yards, as shown in Figure 2, the handling of Figure 5a can be applied: SCL declares the start of the classification. After the last relevant humping process, the newly assembled train needs to be coupled (COU). It is then pulled to a departing track (PUL) for a technical inspection (INS). After the brake check (BRC), the train can depart from the yard (DEP). If the train leaves the yard directly from the classification bowl, the handling can be done as mapped in Figure 5b.

Railcar interchanges cause precedence relations not only *within* trains, but also *between* trains: the start of the classification must begin at the latest by the beginning of the earliest humping process. All other operations of the outbound train must not start before the end of the latest humping process. A feasible schedule of the processes must respect both the precedence relations and the time windows of the processes. The latter are influenced by the selected railcar interchanges.

Since American and European classification yards apply single-stage sorting (Boysen et al., 2012, 2016), we exclude multi-stage sorting in the mapping of processes. The resources must arrive at the process at the starting time of the process and stay until its end. The tours of the resources start and end in the depot. In order to mathematically describe the requirements of each *process*, we transform



(a) Handling for an outbound train departing from the departure bowl

(b) Handling for an outbound train departing from the classification bowl

Figure 5: Example for two different handling requirements of outbound trains

an operational process into one or multiple *requests*. Contrarily to a process, a request requires exactly one resource or a track with a given skill. The used procedure is similar as proposed by Bredström and Rönnqvist (2008): we duplicate each process by the number of qualifications needed for its execution. Since we need to make sure that all requests of one process start at the same time, the requests are connected by synchronization arcs.

Applying this transformation to the handling shown in Figure 4b and an example skill requirement, we perform the transformation from processes to requests. In order to make sure that no other train blocks the track at the same time, we need to guarantee that a blocking request can only be visited after its counterpart – its freeing request. That is why we define so-called vehicle-dependent networks. We obtain a graph as shown in Figure 6.

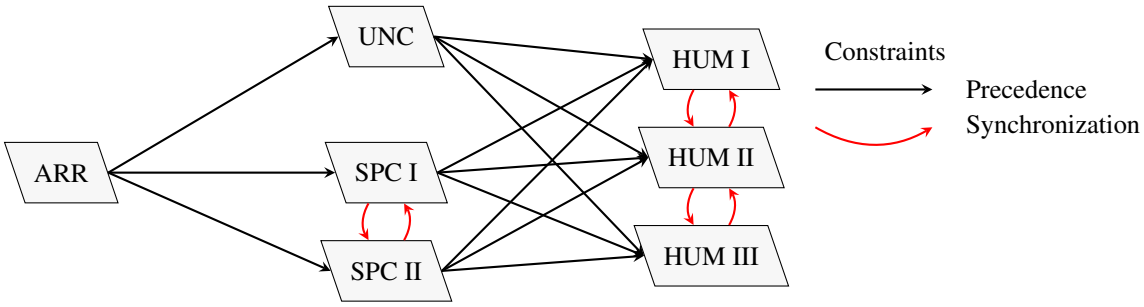


Figure 6: Requests of an inbound train

As certain demand patterns can evoke an overload of the resources in the yard, we allow requests to remain unassigned to local resources. This means that, at a penalty cost, a request can be scheduled and assigned to the "request bank" (Bredström and Rönnqvist, 2008). Ropke and Pisinger (2006a) find that the request bank helps the overall solution process, especially if the minimization of resources is the objective.

3.6. Summary of the problem

Summarizing, the integrated tactical planning problem in classification yards can be defined as follows: given a set of inbound and outbound trains along with information on their timetable, railcars and handling requirements, an infrastructure layout, and information on available resources along with their skills, the goal is to find a feasible operating plan that meets all operational constraints, assigns inbound railcars to outbound trains, schedules all safety and service processes, and assigns them to engines, staff, and infrastructure such that the cost for resources is minimized and the matching between skill requirements and resources is maximized.

4. Mathematical Formulation as a Mixed Integer Linear Program

The described model shares characteristics with both scheduling and routing problems. The scheduling of requests with precedence relations has similarities to job shop scheduling problems (JSPs) (Sec. 3.5). However, the order in which the resources visit several requests is also reminiscent of vehicle routing problems (VRPs). More precisely, the problem can be seen as a special case of the vehicle routing problem with multiple synchronization constraints (VRPMS), which is a VRP where more than one vehicle is used to fulfill a task (Drexl, 2012). Applying the terminology of VRPs, a request can be seen as a *task* and a resource can be seen as a *vehicle*. We refer to Drexl (2012) for a thorough review on VRPMSs.

We now describe the sets and notations that we use. All trains t are collected in set T. Inbound and outbound trains are gathered in the subsets $T^I \subset T$ and $T^O \subset T$, respectively. Each train consists of at least one block $b \in B$ comprising $n_b \geq 1$ railcars. Blocks to destination d are aggregated in set B_d. Analogously, subsets per destination are introduced for trains (e.g., T^O_d for all outbound trains heading to destination d). Outbound train t has a maximum capacity κ_t. The set N contains all requests and the depot $\{0\}$. Subsets of the requests are related to trains (N^t), resources (N^k), and request types (e.g., N^{HUM} for all humping requests). The set S comprises all requests i and j that are synchronized. If request i (e.g., UNC) necessarily precedes request j (e.g., HUM), the pair of requests is included in set A. For each request i, an earliest (s^e_i) and latest possible starting time (s^l_i) is calculated a priori. The set of resources is denoted by R and the applicable subset for request i is $R_i \subset R$. The duration of i is d_i, the transition time of resource k from i to j is denoted u_{ijk}. The costs to utilize resource k is c_k and the penalty cost for assigning a request to the request bank is denoted c^{RB}. As usual, the parameter M denotes a very large value.

We start by introducing the four sets of decision variables. The variable s_{ik} defines the starting time of request i visited by resource k. If k does not visit i, s_{ik} takes value 0. The binary decision variable y_{bt} takes value 1 if the railcars from inbound block b are assembled in outbound train t, and 0 otherwise. The binary variable r_{ijk} indicates whether resource k moves from request i to j or not. Finally, the binary variable z_i takes value 1 if request i is put on the request bank, and 0 otherwise.

The model can be written as follows:

$$\min \sum_{k \in R} c_k \sum_{j \in N} r_{0jk} + c^{RB} \sum_{i \in N} z_i \tag{1}$$

$$\text{s.t.} \quad \sum_{t \in T^O_d} y_{bt} = 1 \qquad \forall d \in D,\ b \in B_d \tag{2}$$

$$\sum_{b \in B_{dt}} n_b \cdot y_{bt} \leq \kappa_t \qquad \forall t \in T^O \tag{3}$$

$$\sum_{k \in R_i} s_{ik} + d_i \leq \sum_{k \in R_j} s_{jk} \qquad \forall (i,\ j) \in A \tag{4}$$

$$\sum_{k \in R_i} s_{ik} = \sum_{k \in R_j} s_{jk} \qquad \forall (i,\ j) \in S \tag{5}$$

$$s^e_i \cdot \left(\sum_{j \in N_k} r_{ijk} - z_i \right) \leq s_{ik} \qquad \forall k \in R,\ i \in N \setminus \{0\} \tag{6}$$

$$s_{ik} \leq s^l_i \cdot \left(\sum_{j \in N_k} r_{ijk} + z_i \right) \qquad \forall k \in R,\ i \in N \setminus \{0\} \tag{7}$$

$$\sum_{k \in R_i} s_{ik} + d_i \leq \sum_{k \in R_j} s_{jk} + (1 - y_{bt}) \cdot M \qquad \forall b \in B,\ i \in N_b,\ t \in T^O,\ j \in \left\{N^t \setminus N^{SCL}\right\} \tag{8}$$

$$\sum_{k \in R_j} s_{jk} + d_j \leq \sum_{k \in R_i} s_{ik} + (1 - y_{bt}) \cdot M \qquad \forall b \in B,\ i \in N_b \cap N^{HUM},\ t \in T^O,\ j \in \left\{N_t \cap N^{SCL}\right\} \tag{9}$$

$$\sum_{j \in N_k} r_{ijk} = \sum_{j \in N_k} r_{jik} \qquad \forall k \in R,\ i \in N_k \tag{10}$$

$$\sum_{\ell \in R_i} s_{i\ell} + d_i + u_{ijk} \leq \sum_{\ell \in R_j} s_{j\ell} + (1 - r_{ijk}) \cdot M \qquad \forall k \in R,\ i \in N \setminus \{0\},\ j \in N \tag{11}$$

$$\sum_{j \in N_k} r_{0jk} = \sum_{i \in N_k} r_{i0k} \qquad \forall k \in R \tag{12}$$

$$\sum_{j \in N_k} r_{0jk} \leq 1 \qquad \forall k \in R \tag{13}$$

$$\sum_{k \in R} \sum_{j \in N_k} r_{ijk} + z_i = 1 \qquad \forall i \in N \tag{14}$$

$$s_{ik} \in \mathbb{N} \qquad \forall i \in N,\ k \in R \tag{15}$$

$$y_{bt} \in \{0,\ 1\} \qquad \forall b \in B,\ t \in T^O \tag{16}$$

$$r_{ijk} \in \{0,\ 1\} \qquad \forall i \in N,\ j \in N,\ k \in R \tag{17}$$

$$z_i \in \{0,\ 1\} \qquad \forall i \in N. \tag{18}$$

The objective (1) is to minimize the number of resources and the costs for unprocessed requests. Constraints (2) and (3) ensure that each inbound block is assigned to exactly one outbound train with the required destination and that the maximum capacity of the outbound trains is respected. Constraints (4) - (9) ensure a feasible schedule for the requests. Constraints (4) make sure that j starts after the end of i if there is a precedence relation. Constraints (5) link the starting times of synchronized requests. Constraints (6) and (7) limit the feasible scheduling time of a request to its time window if it is visited by the relevant resource or if it is set to the request bank. Constraints (8) guarantee that no request (except for the SCL-requests) of the outbound train starts before the last relevant inbound block has been split. Constraints (9) specify that the classification process starts at latest at the beginning of the first humping process. Constraints (10) - (13) guarantee a feasible tour of the resources. A valid flow of the resources is ensured by (10). Constraints (11) guarantee that request j does not start before the end of request i and the resource transition time from i to j if resource k switches between the two requests. Finally, the resources leaving the depot must come back to it (12), each resource can leave the depot at most once (13), and each request must be either visited by a resource or assigned to the request bank (14).

5. Adaptive Large Neighborhood Search

Early experiments show that the presented MILP is intractable for a general-purpose solver applied to instances of realistic size. This brings the need for a tailored solution technique. As described in Section 4, our problem shares characteristics both of routing and scheduling problems. ALNS has been successfully applied on both problem classes (Pisinger and Ropke, 2010) encouraging us to design and implement an ALNS metaheuristic tailored to our problem.

5.1. Basics

ALNS – proposed by Ropke and Pisinger (2006a) for the pickup and delivery problem with time windows (PDPTW) – is an extension of the large neighborhood search (LNS) paradigm introduced by Shaw (1997). Schrimpf et al. (2000) introduced a *ruin and recreate* heuristic similar to LNS. The basic principle of ALNS is that, in each iteration, a solution is destroyed and repaired by appropriate methods. Unlike in LNS, there are several competing destroy and repair operators. Let Ω^- and Ω^+ denote the sets of destroy and repair operators, respectively. The success of the methods is continuously tracked by their scores ρ: the better the methods perform, the more likely they are to be chosen in subsequent iterations. The basic procedure is described in Algorithm 1 (Pisinger and Ropke, 2010).

Data: Feasible solution x
Result: Best found solution x^b
$x^b = x$; $\rho^- = (1, ..., 1)$; $\rho^+ = (1, ..., 1)$;
repeat
  Select destroy and repair methods $d \in \Omega^-$ and $r \in \Omega^+$ using ρ^- and ρ^+;
  $x^t = r(d(x))$;
  if $Accept(x^t, x)$ **then** $x = x^t$;
  if $c(x^t) < c(x^b)$ **then** $x^b = x^t$;
  Update ρ^- and ρ^+;
until *Stopping criterion is met*;
Return x^b;

Algorithm 1: Basic algorithm of the ALNS (Pisinger and Ropke, 2010)

5.2. Framework and settings

The ALNS can be combined with any outer framework that regulates the search, i.e., that decides whether a new solution is accepted or not (Pisinger and Ropke, 2010). Due to its popularity, simplicity and success in other applications (Aarts and van Laarhoven, 1987; Vidal, 1993), we chose simulated annealing (SA) as the outer framework in our solution technique.

In SA, a "temperature" T decides on how likely a solution x' deteriorating the current objective value $c(x)$ will be accepted. The solution is accepted with probability $e^{-(c(x')-c(x))/T}$. As the search progresses, the temperature decreases by the cooling rate c. This allows a rather free movement through the solution space at the beginning, but favors the search to end up in a local optimum.

The start temperature is set as in Pisinger and Ropke (2007): we inspect our initial solution using an adapted objective value without the costs for the request bank. The start temperature is set such that a solution that is w percent worse than the current solution is accepted with probability 0.5: $T = {-w \cdot c(x)}/{\ln 0.5}$.

The new score for destroy methods is calculated as in Cordeau et al. (2010):

$$\rho = \lambda\rho + (1 - \lambda) \cdot \omega, \tag{19}$$

where λ is a decay parameter that decides on how strongly the previous weight is considered, and ω is set as follows. As in Cordeau et al. (2010), we define four weights to evaluate the success of destroy and repair methods: ω_1 if the new solution is a new global best, ω_2 if the new solution is better than the current one, ω_3 if the new solution is accepted, and ω_4 if the new solution is rejected. For a reasonable trade-off between computational time and quality, we additionally consider the computation time of the repair methods (Pisinger and Ropke, 2010). Extending (19), we use

$$\rho_r = \lambda\rho_r + (1 - \lambda) \cdot \omega \cdot \frac{\bar{t}}{\bar{t}_r}, \tag{20}$$

where $\bar{t}$ is the average computation time of all repair methods and $\bar{t}_r$ is that of repair method r. Every n iterations, we reset all weights to their initial values, thus giving a chance to methods that have not been considered previously. Based on reported parameter settings in the literature (Ropke and Pisinger, 2006b; Cordeau et al., 2010) and our own tuning experiments, we use the following parameter values: $w = 0.01$, $c = 0.9995^{25000/\widetilde{n}}$, $\widetilde{n}$ being the number of maximum iterations, $\lambda = 0.9$, $\omega_1 = 100$, $\omega_2 = 40$, $\omega_3 = 10$, $\omega_4 = 1$, and $n = 100$.

5.3. Data structures and methods to ensure feasibility

As the repair methods require fast procedures for testing the feasibility of insertion positions for requests, we put a special emphasis on these methods. Using a direct representation of the solution, efficient data structures have a large impact on the performance of the ALNS.

5.3.1. Graph representation

Let us consider an academic example of one inbound and one (truncated) outbound train as depicted in Figure 7. For the sake of clarity, redundant edges are not displayed, such as additional edges between synchronized requests or between the inbound and outbound train. The graph representing a solution consists of vertices which are the requests of the problem including the depot and edges which express relations between two vertices. The vertices have an assigned starting time and the edges have a weight. The weight corresponds to the maximum amount of time the predecessor can be postponed without causing a temporal overlap with the successor. There are five different kinds of edges. The former three are instance-specific, while the latter two are subject to optimization:

- Precedence edges connect two requests with a predecessor-successor relationship derived from the handling requirements.
- Synchronization edges make sure that two synchronized vertices start at the same time.
- Time constraint edges restrict the time window for fixed requests, e.g., ARR requests.
- Railcar interchange edges connect requests of inbound and outbound trains and vice versa.
- Resource edges are set between two requests if they are contiguously served by one resource.

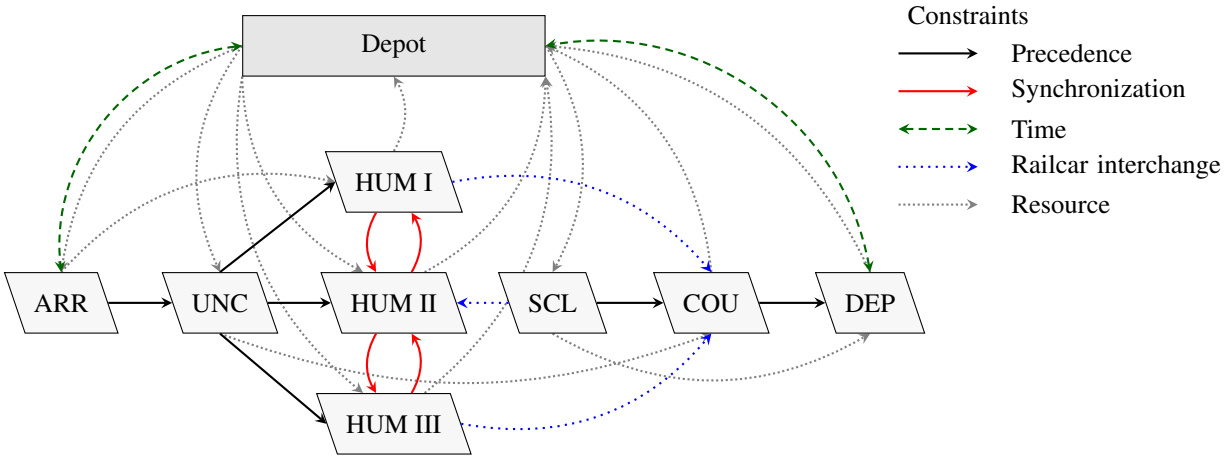


Figure 7: Representation of a solution as a graph

The graph needs to be adapted if the solution is changed. If the scheduling and assignment of a request are removed, the edges containing this information are removed as well. Conversely, edges are inserted into the graph if the solution is repaired, e.g., a request is reinserted into the solution.

Even the academic example in Figure 7, which ignores many additional redundant edges, leads to a complex graph with many interdependent edges. This gives an idea on how quickly the graphs grow for larger instances.

5.3.2. Adaption of the slack principle

The graph presented in Figure 7 helps to evaluate how far we can delay request i without moving neighboring requests by finding the minimum weight of all adjacent edges. Conversely, by finding the minimum weight of all incident edges, we can get the maximum time we can advance the request. However, it might be opportune to allow a delay if the successor and its successors do not cause any infeasibility. Checking this is time-consuming.

This motivates us to adapt the principle of *forward time slack* introduced by Savelsbergh (1991). For the traveling salesman problem with time windows (TSPTW), Savelsbergh proposes to sum all weights of edges between i and j to a constant $F_i^{i,\dots,j}$ called the forward time slack. The request i can be shifted forward by $F_i^{i,\dots,j}$ without causing any infeasibilities. More formally, let l_i be the latest start time of i, D_i the departure time of i, and t_{ij} the travel time between i and j. The forward slack is defined as (Savelsbergh, 1991):

$$F_i^{i,\dots,j} = \min_{i \le k \le j} \left\{ l_k - \left(D_i + \sum_{i \le p < k} t_{p,p+1} \right) \right\}. \tag{21}$$

The idea was developed for problems where nodes are always inserted as soon as possible. In our case, it might be preferable to insert a node at a later position, e.g., it might be opportune to hump a train later than possible to save capacity in the classification bowl. Therefore, we adapt this principle to deal both with forward and backward slack times. We define F_i and B_i as forward and backward slack of i. The forward slack can be calculated by finding the shortest path between i and the depot. Correspondingly, B_i takes the value of the shortest path between the depot and i. If we wish to change the starting time of request i, we can choose any moment in the interval $[s_i - B_i,\ s_i + F_i]$.

5.3.3. Cycle detection

The slack principle described in Section 5.3.2 is not sufficient to guarantee feasibility of an insertion position. In addition, we need to make sure that no cycles are created in the graph. Because it has to be performed for every possible insertion position, the detection of cycles in a graph can be time-consuming. Standard algorithms as the breadth-first search detect cycles in linear time (Courcoubetis et al., 1992; Fisler et al., 2001). We adapt an idea by Grangier et al. (2016), who build on Masson et al. (2013), that enables cycle detection in constant time. If we insert node C on the arc A–B, we need to make sure that there is no path between B and C nor between C and A. As pointed out by Grangier et al. (2016), a cycle can be detected in constant time by the help of a successor matrix (Γ). Maintaining this matrix, e.g., a list of all successors for every request, we can easily detect cycles as a cycle is created if and only if $A \in \Gamma^+(C)$ or $C \in \Gamma^+(B)$.

5.4. Initial solution

To start the ALNS, we require a feasible initial solution. We use a construction heuristic that comprises two parts: first, the railcar interchanges are defined such that a low dwell time is achieved. Second, the requests are scheduled and allocated to applicable resources with the lowest index as

soon as possible. If no applicable resource is found with these rules, the request is put into the request bank.

5.5. Destroy methods

Destroy methods *diversify* and *intensify* the search. Random-driven methods diversify the search. An intensification of the search is yielded by destroying requests that are responsible for a particular badness of the solution (Pisinger and Ropke, 2010). Additionally, typical destroy methods are either *related*, i.e., they destroy similar requests (Pisinger and Ropke, 2010; Shaw, 1998), or *history-based*, that is they try to learn from the previous search progress (Pisinger and Ropke, 2007, 2010).

There are two types of destroy methods considered in this ALNS. *Request removals* (denoted by `dQ`) remove the assignment and scheduling of a request. Unless stated otherwise, the methods are repeated until q requests are destroyed, q being a random number between $\min\{0.1 \cdot |N|, 30\}$ and $\min\{0.4 \cdot |N|, 60\}$ (adapted from Pisinger and Ropke, 2007). Note, that requests related to track allocation are removed in couples, i.e., the blocking and the freeing request, to enable the repair methods to find new solutions despite the restricted graph for tracks.

The second type of destroy methods focuses on *railcar interchanges* and is denoted by `dB`. Here, we destroy assignments from inbound blocks to outbound trains and aim at changing the time windows of the requests, thus allowing to explore new parts of the solution space. Such a destroy method is repeated until b railcar interchanges are destroyed, where b is a random number between $\min\{0.1 \cdot |B|, 4\}$ and $\min\{0.4 \cdot |B|, 8\}$.

5.5.1. General request removals

We start by introducing general request removals.

Random (`dQ1`): we randomly remove a request.

Bowl (`dQ2` / `dQ3` / `dQ4`): For each of the three bowls, we randomly remove a request in this bowl as to bundle the destruction of the solution in one bowl.

Synchronization (`dQ5`): we randomly find a process consisting of multiple requests and remove them all. This increases the freedom to find new starting times for synchronized requests.

Shaw (Shaw, 1997, 1998) (`dQ6`): we remove a random request and then similar requests. We define the relatedness between requests using a weighted sum of the overlap of resources that can be used, starting time, and duration.

Worst (Ropke and Pisinger, 2006a) (`dQ7`): we remove the request increasing the objective value the most.

Historical Request-Pair (Ropke and Pisinger, 2006a) (`dQ8`): we define the relatedness between two requests by the number of times they are served by the same resource in the best $B = 100$ solutions. We remove the most related requests of a random request.

5.5.2. Track-related and resource-related request removals

Some destroy methods focus on the minimization of the number of tracks used by destroying track-related requests.

Random Train (`dQ9`): we randomly choose a train and destroy all its requests.

Track With Few Trains (`dQ10`): we sort the tracks by the number of allocated trains and destroy the requests on the track with the fewest trains.

Other destroy methods intensify the search by focusing on resources (staff, locomotives).

Random Resource (`dQ11`): we randomly choose a resource and remove all its requests.

Resource With Few Requests (`dQ12`): we destroy the resource with the fewest requests.

Resource With Low Productivity (dQ13): we destroy the resource with the lowest productivity defined as the ratio of the time fulfilling requests to the duration between leaving the depot and returning to it.

Historical Node-Pair (Pisinger and Ropke, 2007) (dQ14): we remove arcs that are part of prior solutions with bad objective values.

5.5.3. Railcar interchange removals

These destroy methods remove railcar interchanges to allow new time windows of some requests.

Random (dB1): we randomly remove a railcar interchange.

Evaluated (dB2): we remove a railcar interchange according to an evaluation comprising three weighted terms related to the time between departure and start of classification, the dwell time, and the ratio between blocking time of the classification track before humping and total classification time.

Dwell time (dB3 / dB4): we remove the railcar interchange with the shortest / longest dwell time.

Complete train (dB5 / dB6): we remove all railcar interchanges of an inbound / outbound train.

Heterogeneous interchange (dB7): we remove a railcar interchange that differs from the other interchanges of an inbound train as to condense the number of connections between trains.

Increase in classification tracks (dB8): we remove a railcar interchange that causes an increase in the minimum number of classification tracks.

5.5.4. Slack-related removals

Similar to Christiaens and Vanden Berghe (2018), we want to focus on slack and propose methods that maximize the increase in temporal slack enabling to freely move through the solution space. We use three different methods, two of which focus on request removals (dQ15 - dQ18) and one on railcar interchange removals (dB9 / dB10). All of them work both for forward and backward slack and compare the increase in slack when removing different arcs.

An additional destroy method removes adjacent strings of resource assignment in two variations (dQ19 / dQ20) (Christiaens and Vanden Berghe, 2018) or requests that start in a common time window (dQ21).

5.6. Repair methods

As the destroy methods, we have two types of repair methods in the ALNS: the first type inserts destroyed requests into the solution (denoted by rQ) while the second type repairs railcar interchanges (denoted by rB).

5.6.1. Basic greedy repair

The simplest repair heuristic (rQ1) successively inserts destroyed requests ordered by their indices in the cheapest insertion position. The advantage of this repair method is its speed. Its drawback is that requests with low indices are prioritized independently from the structure of the solution.

5.6.2. Greedy repair

The greedy repair method (rQ2) tries to overcome the drawbacks of the method described in Section 5.6.1. In each iteration, the insertion positions for all destroyed requests are evaluated. The requests which increase the objective value the least are inserted first. The procedure is described in Algorithm 2.

```
Data: Destroyed solution
Result: Repaired solution
Fill list D with destroyed requests;
while |D| > 0 do
    for each i ∈ D do
        if request blocks or unblocks track then Get counterpart ;
        for Each resource k with required qualification do
            Find possible insertion positions ;
            for Each possible insertion position do
                Save i, k, arc, Δ objective with and without request ;
    if At least one possible insertion position found then
        Insert i* at cheapest position;
    else
        Set request with lowest index i* to request bank;
    Update graph and slacks ;
    Remove i* from D;
```

Algorithm 2: Greedy repair.

5.6.3. k-regret repair

The k-regret repair method (`rQ3`) works similar to the Greedy repair heuristic (Potvin and Rousseau, 1993). The only difference is the choice of the best option: we choose the option with the highest difference in the objective value of the kth-best option and the best option. That is how we try to detect early if good insertion positions for a given request get scarce.

5.6.4. Resource-ranked greedy repair

The resource-ranked greedy repair method (`rQ4`) focuses on bottlenecks in the resource pool. It consists of a ranking and a repair method. First, the ranking method dynamically adjusts the importance of the destroyed requests. Second, a subset of prioritized requests is inserted with the k-regret repair method.

The following ranking methods are available:

- *by slack:* used resources are sorted by the average slack of the served requests. The lower the slack, the higher the priority.
- *by active time:* the more time a resource requires for its requests, the higher it is ranked.
- *by the ratio of necessary to available time:* the higher the ratio of necessary time for the additional resources to available time of the resources, the higher the resource is ranked.

The result is a ranking of resources. The subset of destroyed requests inserted by the repair method is determined as follows: all requests that require the highest ranked and unprocessed resource.

5.6.5. 2-stage railcar interchange repair

If a destroy method removes railcar interchanges, the repair methods described before cannot be applied. Therefore, we need specialized repair methods that are described in the following.

We design a repair method that repairs the railcar interchanges but which, at the same time, has the necessary freedom to find new railcar interchanges. Therefore, we propose a 2-stage repair heuristic (Algorithm 3). It first neglects constraints related to the resource routing. Consequently, the repair of railcar interchanges might cause infeasible routing and scheduling constraints, which forces the

algorithm to destroy requests with an infeasible schedule (e.g., the schedule of requests of inbound and outbound trains contradicts a newly introduced railcar interchange), and then – in a second stage – calls a repair method to insert the newly destroyed requests.

```
Data: Destroyed solution
Result: Repaired solution
for Each unassigned block b do
    Find possible outbound trains (check destination, capacity, and minimum dwell time) ;
    for Each possible outbound train do
        Evaluate railcar interchange ;
    Choose outbound train with best evaluation and adapt block assignment;
    for Each request of inbound and outbound train do
        if Current starting time violates new constraints then
            Set time to earliest (inbound train, SCL) or latest begin (else) ;
            Destroy request ;
Call k-Regret Repair ;
```

Algorithm 3: 2-stage railcar interchange repair

The algorithm is implemented in several variants with respect to the evaluation of the railcar interchanges. The variants are as follows:

- assign railcar interchanges randomly (`rB1`),
- choose railcar interchanges as soon / late as possible (`rB2` / `rB3`),
- assign interchanges that reduce the number of connections between inbound and outbound trains (`rB4`), and
- choose the railcar interchanges that produce "good" time windows for the classification tracks (`rB5`).

Depending on the structure of the solution, this repair method risks to either destroy few requests, making it difficult to improve current solutions, or many requests, causing high computational times and possibly poor results.

6. Computational results

6.1. Test instances

For the computational results, we consider two sets of instances (denoted by I^1 and I^2). The instances vary in the size of the yard (I^1), the number (I^2) and skills (I^1 and I^2) of the resources, and the number (I^1 and I^2) and type (I^1) of trains. The parameters and their chosen value can be found in Table 2.

The sets of resources vary in number and in the share of cross-trained personnel. We consider three levels of cross-training: none (each resource is trained in one skill), medium (each resource has 2-3 skills), and high (each staff is trained in every qualification, each locomotive has full technical equipment).

For the instances of set I^1, the number of available locomotives and staff is fixed, while it varies for the other instances (see Table 2 for the chosen values). For the infrastructure elements, we consider two yard sizes: a small yard with two receiving tracks, eight classification tracks, and four departure tracks, as well as a medium yard being equipped with double the infrastructure elements.

The number of inbound trains (γ) equals the number of outbound trains. The arrival and departure times of the trains are randomly drawn from the interval $[\underline{t}^{arr};\ \bar{t}^{arr}]$ and $[\underline{t}^{dep};\ \bar{t}^{dep}]$, respectively. Each inbound train contains δ blocks which have a random destination ϵ. Each outbound train has a capacity κ. The number of railcars per inbound block is randomly drawn out of the interval $[1,\ \bar{n}]$, where $\bar{n}$ is calculated as $\lfloor \zeta \cdot \kappa / \delta \rfloor$, where ζ is the ratio of inbound railcars and outbound capacity. The four basic handling procedures for both inbound and outbound trains were introduced in Figures 4 and 5, respectively. The parameter η defines the share of trains with more complex process chains resulting in a higher number of requests.

The parameters used in both sets of instances can be found in Table 2. While the arrival and departure times in the set of instances I^1 are set such that each inbound train can possibly have connections to each outbound train, the instances in I^2 consider overlapping time windows of inbound and outbound trains.

The train characteristics and their handling requirements are the base for the generation of the processes and requests. The earliest and latest possible begins are derived by the critical path method. The durations of the processes θ take the values $\{5; 30; 20; 15; 1; 30; 25; 60; 10; 5\}$ for the processes {ARR, UNC, SPC, HUM, SCL, COU, PUL, INS, BRC, DEP}. These values are estimated and similar values can be found in Bohlin et al. (2011a). Full information on the instances is available on `https://github.com/MoritzRuf/IntegratedCYPlanningInstances/`.

Parameter		Values (I^1)	Values (I^2)
#R	Number of resources per skill category	3	$\{1; 2; 3; 5\}$
#L	Number of locomotives per equipment	2	$\{1; 2; 4\}$
α	Multi-skilling of staff	$\{0; 1; 2\}$	$\{0; 1; 2\}$
β	Yard size (number of RT / CT / DT)	$\{2/8/4; 4/16/8\}$	$\{4/16/8\}$
γ	Number of inbound and outbound trains	$\{5; 10; 20\}$	$\{5; 10; 20\}$
δ	Number of blocks per inbound train	4	4
κ_t	Maximum number of railcars per outbound train	40	40
$\underline{t}^{arr} \mid \bar{t}^{arr}$	Earliest and latest possible arrival times	10 | 610	10 | 810
$\underline{t}^{dep} \mid \bar{t}^{dep}$	Earliest and latest possible departure times	810 | 1410	610 |1410
ϵ	Number of destinations	$\{1; 2; 5\}$	$\{2; 5\}$
η	Share of complex process chains	$\{0; 0.5; 1\}$	0.5
ζ	Ratio inbound railcars to outbound capacity	$\{0.5; 0.75; 1\}$	0.75
	# instances	486	216

Table 2: Overview of parameters of the test instances and their values

6.2. Setup

We run the computational experiments on a MacBookPro with a 2.4 GHz processor and 8 GB of RAM. The ALNS has been implemented in C++. We solve the instances described in Section 6.1. Due to the train makeup problem, six instances of set I^1 and 54 instances of set I^2 turn out to be (mathematically proven) infeasible. We present figures that relate to the remaining 480 (I^1) and 162 (I^2) instances.

We report computational results with respect to both performance and solution quality. We run five experiments differing in the stopping criterion (10,000, 25,000, 50,000, 100,000, and 150,000 iterations, referred to as 10k, 25k, 50k, 100k, and 150k, resp.). First, we analyze the behavior of the algorithm with 10k experiments. Second, we study the changes when increasing the number of iterations. Last, we benchmark the results of the ALNS by comparing them against optimal solutions.

6.3. 10k results

We now report computational results of the ALNS with 10,000 iterations. First, we provide some figures on the computational time of the ALNS. Second, we have a closer look at how likely the different destroy and repair methods are chosen.

The overall average CPU time for the experiments with 10k iterations on the instances I^1 is 110 s (20 s, 75 s, 234 s for 5 trains, 10 trains, and 20 trains, respectively) and for the instances of set I^2 125 s (21 s, 67 s, 229 s for 5 trains, 10 trains, and 20 trains, respectively). For the instances of set I^1, 26.2% of the time is needed for the destroy methods, 62.9% for the repair methods, and 10.9% for overhead functions on average. The values for the remaining instances are similar. A scatterplot which depicts the relation between the number of requests and computational time can be found in Figure 8.

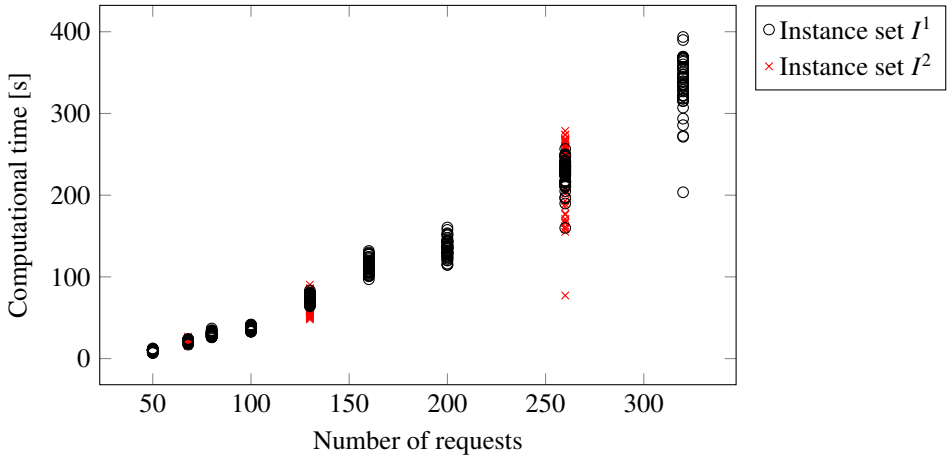


Figure 8: Computational time of the ALNS (10k) depending on request number

In total over the 642 instances, the ALNS could find 12,177 (0.19% of all iterations, resp.) new best solutions, it could improve the current solution 3,230 times (0.05%, resp.), and provided 2.50 M (38.96%, resp.) accepted solutions. A total of 3.90 M (60.81%, resp.) solutions were rejected.

We now analyze the frequency at which the destroy methods were called. From the frequency, one can draw conclusions on how the operators have been evaluated during the solution process. Figure 9 displays the call frequency divided by type of destroy method.

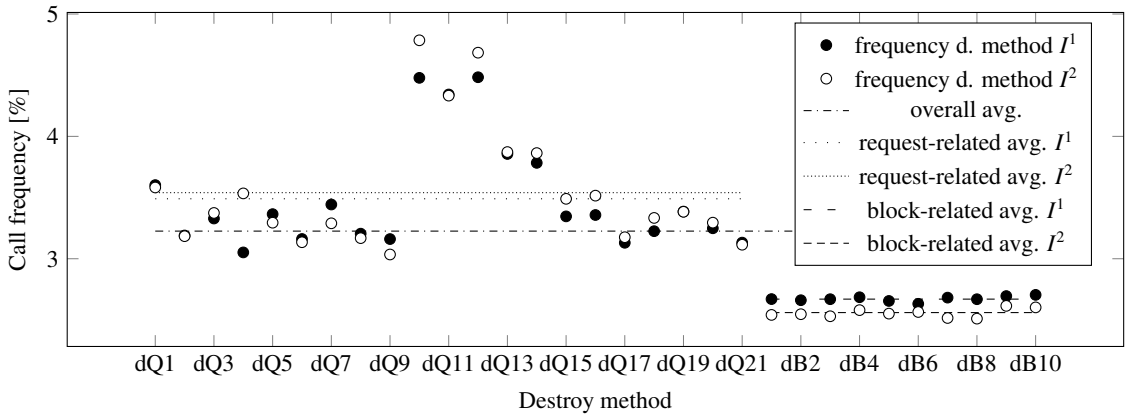


Figure 9: Call frequency of the destroy methods (10k experiments)

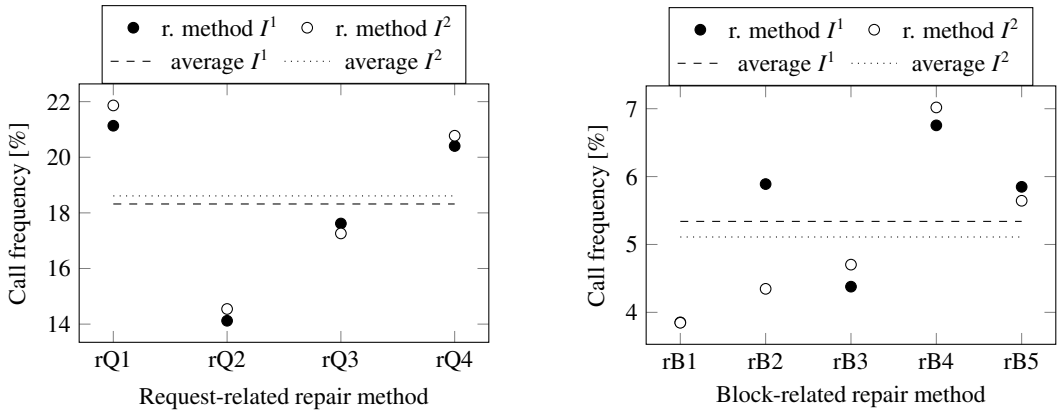
If all methods had the same score, each method would be called in 3.23% of the iterations. We have a total of 31 destroy methods, of which 21 (67.74%) are request-related. In total, 73.28% of the time, request-related methods are called for instances of set I^1, indicating that they are somewhat more successful than block-related ones. For the instances of set I^2, the share is 74.44%. Due to the restricted possible connections between inbound and outbound trains, the block-related methods have a smaller success.

Focusing on the request-related methods, the average call frequency is 3.49% (3.54%) for instances of set I^1 (I^2) and varies between 3.03% and 4.79% The method that is called the least (3.05%) for I^1 focuses on requests in the departing bowl (`dQ4`). In a third of the instances of this set, no requests occur on departing tracks since all trains depart from the classification track, making this method useless for these cases. For the I^2 instances, the destroy method `dQ9` removing the requests of a random train is called least.

Both destroy methods that are called most frequently, (`dQ10` and `dQ12`, resp., both 4.48% for I^1 and 4.68% and 4.79% for I^2, resp.), exploit information on the resources: on their number of requests (`dQ12`) and the number of trains on tracks (`dQ10`). These intensifying methods are closely related to the objective function. Similarly, the destroy methods removing a random resource (`dQ11`, 4.34% and 4.33% for I^1 and I^2, resp.) intensify the search and is the third most successful request-related destroy method. Other methods that are called more frequently than the average focus on the resource productivity (`dQ13`, 3.86% / 3.87%), exploit historical node-pair information (`dQ14`, 3.78% / 3.86%), and randomly destroy requests (`dQ1`, 3.60% / 3.58%).

Compared to the request removals, the variance in call frequency is much lower for block-related methods (coefficient of variation (CV) 12.79% / 14.08% and 0.78% / 1.37% for request and block removals, resp.). The frequency ranges between 2.63% and 2.70% for I^1 and between 2.51% and 2.61% for I^2.

We now focus on repair methods. Recall that depending on the destroy methods, only a subset of the repair methods may be called. Figure 10 displays the call frequency both for request-related (Fig. 10a) and block-related (Fig. 10b) repair methods.



(a) Call frequency request-related repair methods

(b) Call frequency block-related repair methods

Figure 10: Call frequency of the repair methods (10k experiments)

From the request-related repair methods, the basic greedy repair (`rQ1`) and the resource-ranked greedy repair (`rQ4`) are called most frequently (21.1%/21.9% and 20.4%/20.8%, resp.). Both methods have in common that they are faster than the other two methods since they only treat a subset of destroyed requests in each iteration. The greedy repair method (`rQ2`) falls back behind the other

methods (14.1%/14.5%) while the k-regret heuristic (rQ3) works considerably better (17.6%/17.3%).

For the block-related repair methods, the repair method minimizing the number of connecting trains (rB4) is called most frequently (6.8%/7.0%). The other repair methods performing above average assign the railcars as soon as possible to outbound trains (rB2, for set I^1 5.9%) and such that the lower bound for classification tracks is minimized (rB5, 5.8%/5.6%). The repair methods assigning railcars to outbound trains as late as possible (rB3, 4.4%/4.7%) and randomly (rB1, 3.9%/3.8%) are called least frequently.

The computational time both of the destroy and the overhead methods are mainly driven by the number of requests, e.g., by the number of trains and the share of complicated process chains. The repair methods are additionally affected by the yard size and the level of cross-training.

6.4. Experiments with a higher number of iterations

Next, we report results of experiments with an increased number of iterations. We analyze the impact on the computational time and the quality of the solutions. Additionally, we check whether the success of the methods is affected by the number of iterations. The overall average CPU time of the experiments for I^1/I^2 rises to 208 s/271 s, 566 s/554 s, 1186 s/1101 s, and 1826 s/2110 s for 25k, 50k, 100k, and 150k iterations, respectively. The increase in time roughly corresponds to the increase in iterations (factor 1.9/2.2, 5.1/4.4, 10.7/8.8, and 16.5/16.9). The share of chosen destroy and repair methods is close to the one reported for the experiments with 10k iterations. The CV of the call frequency of all destroy methods is on average 0.46%/0.44%. The share of request-related slightly increases with a higher number of iterations (73.28%/74.44%, 73.54%/74.43%, 73.54%/74.54%, 73.61%/74.61%, and 73.65%/74.61% for 10k, 25k, 50k, 100k, and 150k resp.). The variability being overall very low, we do not report the changes in detail.

We now investigate the changes in the number of solutions found per evaluation group. The share of iterations with a given evaluation can be found in Table 3 for all computational experiments. Compared to the baseline set by the experiments with 10k, the number of best found solutions increases by 1.2%/2.0%, 5.3%/8.5%, 6.2%/9.6%, and 7.2%/10.5% for the experiments with 25, 50k, 100k, and 150k, respectively. With increasing number of iterations, the share of solutions improving the current solution (ω_2) slightly rises. Additionally, we can observe the share of accepted solutions drops from 38.66%/39.84% to 35.62%/36.65% while the number of rejected solutions rises. As a first indicator, we can conclude that even with a relatively low number of iterations (10,000), a high share of the solutions can be found. Comparing the two sets of instances, we find that increasing the number of iterations has a larger impact on the solutions of the second set of instances (I^2).

	# iterations	Best sol. (ω_1)		Improving sol. (ω_2)		Accepted sol. (ω_3)		Rejected sol. (ω_4)	
I^1	10k	8 923	0.19%	2 106	0.04%	1 855 460	38.66%	2 933 511	61.11%
	25k	9 027	0.08%	6 118	0.05%	4 494 535	37.45%	7 490 320	62.42%
	50k	9 392	0.04%	12 772	0.05%	8 793 560	36.64%	15 184 276	63.27%
	100k	9 473	0.02%	25 842	0.05%	17 260 802	35.96%	30 703 883	63.97%
	150k	9 564	0.01%	38 715	0.05%	25 647 226	35.62%	46 304 495	64.31%
I^2	10k	3 254	0.20%	1 124	0.07%	645 462	39.84%	970 160	59.89%
	25k	3 319	0.08%	3 296	0.08%	1 558 707	38.49%	2 484 678	61.35%
	50k	3 530	0.04%	6 606	0.08%	3 059 754	37.78%	5 030 110	62.10%
	100k	3 566	0.02%	12 909	0.08%	6 001 382	37.05%	10 182 143	62.85%
	150k	3 595	0.01%	19 686	0.08%	8 906 147	36.65%	15 370 572	63.25%

Table 3: Average number and share of solutions per evaluation group

We now investigate the change in the objective value, i.e., we are interested in quantifying the change in solution quality by an increase in the number of iterations. Compared to the average objective function of 290/208.8 for the 10k experiments, the value can be lessened by 8.1/6.4 (2.8%/3.0%), 12.6/9.6 (4.4%/4.6%), 14/12.6 (4.8%/6.0%), and 15.6/14.0 (5.4%/6.7%) for the experiments with 25k, 50k, 100k, and 150k, respectively. The median value of the savings is 0 for all experiments, indicating that already 10k iterations seem to set a good baseline in terms of solution quality.

Table 4 shows the number of instances for which the objective value improves, stays the same or deteriorates. As an example, for 94 (19.6%) instances of set I^1, the experiments with 100k find a better solution than the one found in the 50k iterations. For 39 (8.1%) instances, worse solutions were found compared to the experiments with 50k iterations. Note that due to the stochasticity in the solution technique, a higher number of iterations does not guarantee to find at least as good solutions. For the remaining 347 instances (72.3%), the same objective value was found. It can be observed that with increasing number of iterations, the number of improved solutions declines. Comparing the share of instances that can be improved when increasing the number of iterations, we can observe that the share is lower for instances of set I^1 than for the set I^2. Hence, we could conclude that the instances of the latter set are harder to solve. This is linked with that the fact that a higher share of infeasible solutions of set I^2 contains 5 trains.

	Instances of set I^1								Instances of set I^2							
Experiments	Obj. value		Better		Same		Worse		Obj. value		Better		Same		Worse	
25k vs. 10k	281.9	290.0	155	32.3%	296	61.7%	29	6.0%	202.4	208.8	61	37.7%	91	56.2%	10	6.2%
50k vs. 10k	277.4	290.0	203	42.3%	259	54.0%	18	3.8%	199.2	208.8	77	47.5%	79	48.8%	6	3.7%
50k vs. 25k	277.4	281.9	105	21.9%	341	71.0%	34	7.1%	199.2	202.4	45	27.8%	105	64.8%	12	7.4%
100k vs. 10k	276.0	290.0	256	53.3%	217	45.2%	7	1.5%	196.2	208.8	93	57.4%	69	42.6%	0	0.0%
100k vs. 25k	276.0	281.9	141	29.4%	319	66.5%	20	4.2%	196.2	202.4	65	40.1%	92	56.8%	5	3.1%
100k vs. 50k	276.0	277.4	94	19.6%	347	72.3%	39	8.1%	196.2	199.2	47	29.0%	103	63.6%	12	7.4%
150k vs. 10k	274.4	290.0	229	47.7%	243	50.6%	8	1.7%	194.8	208.8	89	54.9%	73	45.1%	0	0.0%
150k vs. 25k	274.4	281.9	159	33.1%	307	64.0%	14	2.9%	194.8	202.4	67	41.4%	90	55.6%	5	3.1%
150k vs. 50k	274.4	277.4	113	23.5%	348	72.5%	19	4.0%	194.8	199.2	47	29.0%	108	66.7%	7	4.3%
150k vs. 100k	274.4	276.0	78	16.3%	364	75.8%	38	7.9%	194.8	196.2	27	16.7%	117	72.2%	18	11.1%

Table 4: Comparison of the objective values depending on number of iterations and sets of instances

We now want to better understand both the level of the improvements and the structure of the instances for which an increase of iterations produces new best solutions. Tables 5 and 6 show the share of instances that can be improved and the average change related to the instance attributes. It can be seen, as can be expected, that the possible improvements are on average higher for larger instances and thus that mainly the attribute number of trains governs this share. Comparing the results for 25k with those of 10k for the instances of set I^1 (Table 5), 8.9% of the solution of all instances with 5 trains can be improved, while the share rises to 56.2% for instances with 20 trains. Comparing 150k with 100k, there is still a considerable share of 34.6% of the instances with 20 trains that can be improved. The average change in objective value strongly decreases when comparing the results of the experiments with a higher number of iterations. For the instances with 20 trains, it drops from 20.6 (25k vs. 10k) to 4.2 (150k vs. 100k).

The observations for the I^2 instances are similar to those described above. For the instances having very few resources, the largest improvements can be found. This relates to the fact that requests that do not need to be set to the request bank have a rather large impact on the objective function value.

		# Trains		Complic. Trains		Destinations		Cap ratio		Yard size		Multi-skilling	
25k	a	8.9%	-0.8	20.0%	-6.6	37.7%	-8.3	25.3%	-5.6	33.3%	-11.6	30.9%	-8.0
vs.	b	31.3%	-2.6	34.8%	-9.2	33.5%	-6.6	34.0%	-6.8	31.3%	-4.5	35.0%	-7.4
10k	c	56.2%	-20.6	42.1%	-8.4	25.5%	-9.2	37.8%	-11.9			31.0%	-8.8
50k	a	2.5%	-0.1	16.9%	-3.6	23.5%	-4.7	19.8%	-4.1	23.3%	-7.1	22.2%	-5.3
vs.	b	18.8%	-1.4	21.7%	-3.5	25.5%	-4.8	20.4%	-3.7	20.4%	-2.0	21.9%	-4.3
25k	c	43.8%	-12.0	27.0%	-6.6	16.6%	-4.2	25.6%	-5.9			21.5%	-4.1
100k	a	3.8%	-0.4	12.5%	0.3	19.8%	0.1	14.8%	-0.1	17.5%	-1.0	16.7%	-1.7
vs.	b	14.4%	-0.8	23.0%	-2.4	24.2%	-2.4	18.5%	-1.0	21.7%	-1.8	20.6%	-1.1
50k	c	40.1%	-2.9	23.3%	-2.1	14.7%	-1.8	25.6%	-3.1			21.5%	-1.3
150k	a	0.6%	0.1	14.4%	-2.4	16.1%	-2.0	14.8%	-0.9	16.3%	-2.4	14.8%	0.2
vs.	b	13.1%	-0.7	11.2%	-1.1	16.8%	-0.8	14.8%	-2.3	16.3%	-0.9	18.1%	-3.6
100k	c	34.6%	-4.2	23.3%	-1.4	15.9%	-2.1	19.2%	-1.7			15.8%	-1.5

Table 5: Share of instances that can be improved and average change in objective value by instance attributes (instances I^1)

Figure 11 shows the average objective value of all instances per number of iterations along with the share of instances that either have found the best known solutions of the ALNS or are within a given range of gap with respect to the best known solution. The average objective value steadily decreases from 290 to 274.4 for instances of set I^1 and from 208.8 to 194.8 for the instances of set I^2, respectively. Being still considerable until 50k, the curve flattens for the experiments with 100k and 150k.

For the instances of set I^1 (Fig. 11a), we find for the experiments with 10k iterations, 47.7% of the solutions correspond to the best found solutions in all experiments. Another 37.7% are within a range of 10% gap and 2.9% of the instances have a gap higher than 20%. For the experiments with 150k iterations, 87.9% of the best known solutions can be found. This means that 12.1% of the best found solutions in experiments with a lower number of iterations could not be found again. Except for four instances, all other solutions are within a range of 10% gap.

The graph for the I^2 instances (Fig. 11b) shows that for 37.7% of the instances, the solution found in the 10k experiments corresponds to the best found solution in all experiments. This is aligned with the findings of Table 4 indicating that these instances are harder to solve than the instances of set I^1.

6.5. Quality benchmarking

We now assess the ALNS in terms of solution quality. To this end, we compare the best found solution in all experiments to the results of a general-purpose solver (Gurobi Optimization, 2019) and see whether it can improve the solution or prove optimality. If no optimum can be found, the reported lower bound refers to the best bounds found within 60 minutes. Since we aim at finding as many optimal solutions as possible, we increase the number of optimal solutions found by applying a second general-purpose solver (IBM ILOG CPLEX, 2015) as well with a higher time limit. The solvers are provided with the best solutions found by the ALNS as a warm start.

Only for 14/4 of the 480/162 (3.5%/2.5%) instances, the solver could improve the solution of the ALNS in the given time limit. However, even though additionally provided cuts help to improve the share of optimally solved instances, a significant part of the instances still cannot be solved to optimality. We therefore divide the quality assessment into two parts: after giving an idea about the share that can be solved to optimality, we first analyze the results for the instances for which we know the optimal solution. Second, we take the best found lower bound as the benchmark.

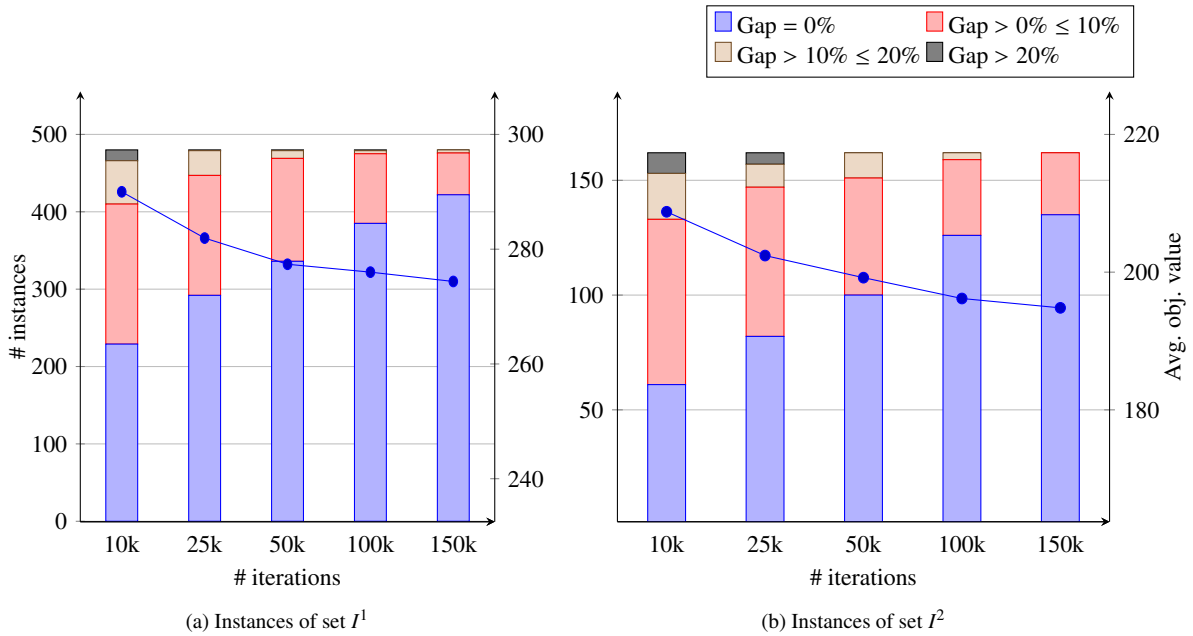
		# Trains		# Resources		# Locomotives		Multi-skilling		Destinations	
25k vs. 10k	a	2.6%	0.0	52.5%	-15.5	32.1%	-5.5	48.1%	-9.1	39.4%	-4.6
	b	33.3%	-2.6	40.5%	-4.3	46.3%	-8.1	30.9%	-6.4	34.9%	-9.0
	c	60.9%	-12.9	34.1%	-3.9	34.6%	-5.4	34.0%	-3.6		
	d			22.2%	-1.7						
50k vs. 25k	a	0.0%	0.5	27.5%	-6.0	26.8%	-2.7	24.1%	-2.0	28.3%	-3.1
	b	22.2%	-1.9	31.0%	-2.1	24.1%	-2.6	32.7%	-4.7	27.0%	-3.3
	c	47.8%	-6.4	22.7%	-1.6	32.7%	-4.4	26.4%	-2.8		
	d			30.6%	-3.3						
100k vs. 50k	a	10.3%	-1.0	35.0%	-5.3	25.0%	-1.8	22.2%	-1.9	27.3%	-2.1
	b	16.7%	-1.1	26.2%	-1.9	24.1%	-2.6	30.9%	-4.5	31.7%	-4.4
	c	49.3%	-5.7	25.0%	-2.0	38.5%	-4.8	34.0%	-2.6		
	d			30.6%	-3.1						
150k vs. 100k	a	0.0%	0.0	30.0%	-5.0	19.6%	-2.7	13.0%	-1.1	16.2%	-1.3
	b	3.7%	0.4	14.3%	-0.7	18.5%	-1.9	14.5%	-1.6	17.5%	-1.4
	c	36.2%	-3.5	11.4%	0.0	11.5%	0.6	22.6%	-1.3		
	d			11.1%	0.3						

Table 6: Share of instances that can be improved and average change in objective value by instance attributes (instances I^2)

In total, the optimum of 189/56 instances can be found and proven. This corresponds to a share of 39.4%/34.6%. The difficulty and size of the problem is mainly driven by the number of requests, i.e., by the number of trains and the share of trains with a more complicated process chain. In addition, the level of multi-skilling has a substantial impact on the number of optimally solved instances. For single-skilled resources, for 52.5%/55.6% of the instances, the optimal solution can be found, while the share drops to 25.9%/20.8% for fully multi-skilled locomotives and staff. For the I^2 instances, the share of optimally solved instances decreases by an increase in the number of locomotives (from 39.3% to 28.8%) and resources (from 37.5% to 33.3%), respectively. For the instances with two destinations, a share of 40.4% can be optimally solved, while for only 25.4% of the instances with 5 destinations, an optimum can be found.

We now discuss the optimality gap of the the results of the ALNS for the instances we know the optimal objective value for. On average, the best found solution of the ALNS in all experiments has an optimality gap of 0.34% for instances of set I^1 and 0.49% for the remaining instances, respectively. Depending on the number of iterations, the average optimality gap varies between 1.72%/2.71% (10k) and 0.57%/0.86% (100k/150k). For the 150k results of the instances of set I^1, the average optimality gap is 0.59%. While the overall average objective function for the 150k results is lower than for 100k, the 100k experiments yield slightly better results for those instances that could be solved to optimality. For the best found solutions, the optimal solution can be found by the ALNS in 95.2%/94.6% of the cases (varying between 78.3%/62.5% for 10k and 92.6%/85.7% for 100k, and 92.1%/87.5% for 150k resp.). The absolute value between the objective value of the best found solution by the ALNS and the optimum is in no case higher than 10. This corresponds to the cost of a resource and is the smallest possible change in the objective function. The share of instances having more than 8% of optimality gap is 2.1%/3.6% for the best found solutions and 11.1%/8.9%, 6.9%/5.4%, 5.3%/8.9%, 3.2%/3.6%, and 3.7%/3.6% for the other experiments, respectively.

An attribute-sensitive analysis of the computational results is provided in Figure 12. The observations can be summarized as follows: an increase in the number of iterations has a high impact on the quality for the instances with 10 trains. The same holds true for an increasing share of trains with complicated process chains. The average gap for instances with a higher number of destinations is



(a) Instances of set I^1

(b) Instances of set I^2

Figure 11: Number of instances for each stopping criterion by gap with respect to best known solutions by all experiments

surprisingly low. A high number of destinations decreases the solution space of the train makeup problem, indicating that this subproblem might be better investigated by extensions of the ALNS.

We now apply the quality assessments on the best lower bounds that the solver could find. This is a worst case assessment of the solution quality of the ALNS. The average gap between best solution of the ALNS and the highest lower bound is 11.6% and 18.4% for the instances of set I^1 and I^2, respectively, varying between 0.9%/1.2%, 12.2%/12.6%, and 21.5%/32.6% for 5, 10, and 20 trains, respectively. Its distribution is displayed in the histogram in Figure 13. For 77.3%/51.2% of the instances, the optimality gap to the lower bound is less than 20%.

7. Conclusion and further research

This paper introduces and solves an optimization problem which is important for integrated planning in railroad classification yards. While most studies in the OR literature focus on particular subproblems of the planning problem, we focus on an integrated setting. The studied problem incorporates all four subproblems defined by (Boysen et al., 2016) and encloses the *scheduling and assignment problem*, which is crucial for the everyday challenges of practitioners but has seen very little attention in the literature so far. The optimization problem assigns inbound railcars to outbound trains, decides on the humping sequence, defines the track allocation, schedules all arising processes, and allocates personnel and locomotives to the tasks. To the best of our knowledge, we are the first to provide an integrated MILP formulation for this problem.

Since the model appears intractable for general-purpose solvers for instances of practical size, we develop a tailored ALNS heuristic. An extensive numerical assessment with an instance pool containing problems of different sizes and structures proves that the ALNS yields very good solutions in reasonable time. The average optimality gap for solutions of which the optimal solutions are known is below 0.5%. Hence, we provide a powerful algorithm to facilitate complex decisions arising in classification yards every day.

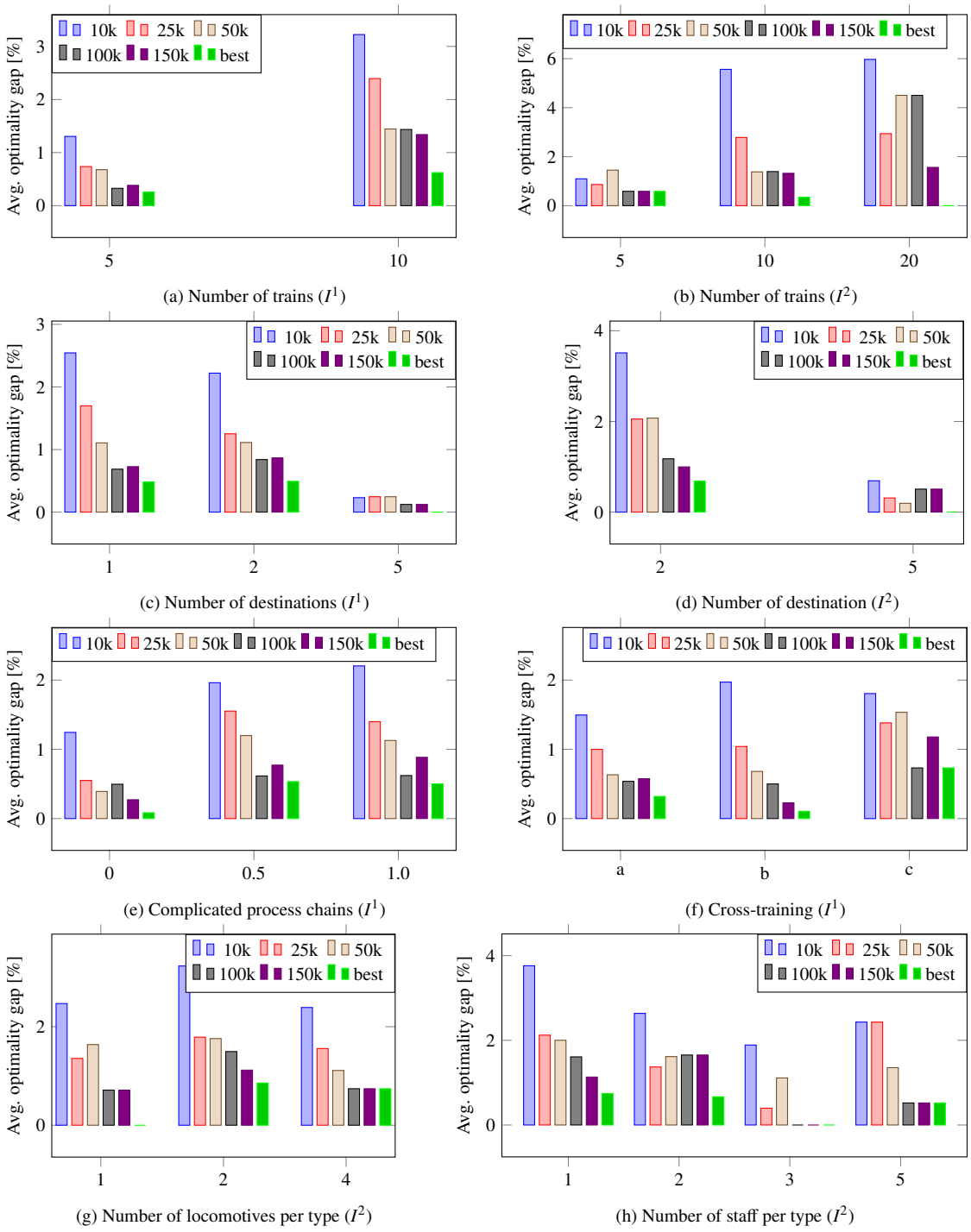


Figure 12: Avgerage optimality gap of the best ALNS solutions per attribute

Further research should be dedicated to the decomposition of the mathematical problem to allow finding optimal solutions to large instances. Another question relates to the benefits of an integrated model as presented in this paper. It would be interesting to compare the results of this model to a sequential approach as is commonly used in practice.

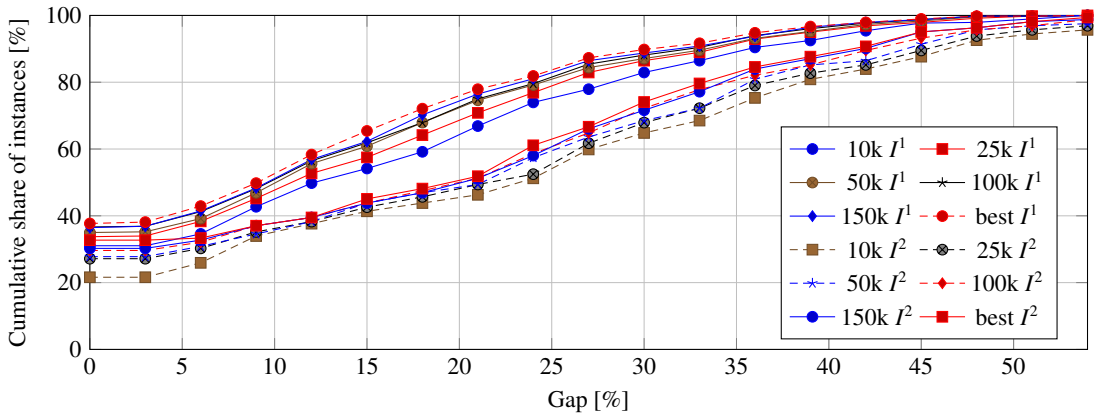


Figure 13: Distribution of gap between best solution of ALNS and best lower bound of solver

Acknowledgments

We gratefully acknowledge the support of the German Academic Exchange Service (DAAD) for the first author's research stay at CIRRELT (Montreal, Canada).

References

Aarts, E. H. L., & van Laarhoven, P. J. M. (1987). Simulated annealing: a pedestrian review of the theory and some applications. *Pattern Recognition Theory and Applications*, *F30*, 179–192. doi:10.1007/978-3-642-83069-3_15.

Ahuja, R. K., Jha, K. C., & Liu, J. (2007). Solving real-life railroad blocking problems. *Interfaces*, *37*, 404–419. doi:10.1287/inte.1070.0295.

Assad, A. A. (1980). Modelling of rail networks: toward a routing/makeup model. *Transportation Research Part B*, *14B*, 101–114. doi:10.1016/0191-2615(80)90036-3.

Assad, A. A. (1981). Analytical models in rail transportation: An annotated bibliography. *INFOR: Information Systems and Operational Research*, *19*, 59–80. doi:10.1080/03155986.1981.11731807.

Bektas, T., Crainic, T. G., & Morency, V. (2009). Improving the performance of rail yards through dynamic reassignment of empty cars. *Transportation Research Part C*, *17C*, 259–273. doi:10.1016/j.trc.2008.11.003.

Belošević, I., & Ivić, M. (2018). Variable neighborhood search for multistage train classification at strategic planning level. *Computer-Aided Civil and Infrastructure Engineering*, *33*, 220–242. doi:10.1111/mice.12304.

Belošević, I., Ivić, M., Kosijer, M., Pavlović, N., & Aćimović, S. (2015). Challenges in the railway yards layout designing regarding the implementation of intermodal technologies. In *2nd Logistics International Conference, Belgrade, Serbia* (pp. 62–67).

Bodin, L. D., Golden, B. L., & Schuster, A. D. (1980). A model for the blocking of trains. *Transportation Research Part B*, *14B*, 115–120. doi:10.1016/0191-2615(80)90037-5.

Bohlin, M., Dahms, F., Flier, H., & Gestrelius, S. (2012). Optimal freight train classification using column generation. In D. Delling, & L. Liberti (Eds.), *12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12)* (pp. 10–22). doi:10.4230/OASIcs.ATMOS.2012.10.

Bohlin, M., Flier, H., Maue, J., & Mihalák, M. (2011a). Hump yard track allocation with temporary car storage. In *The 4th International Seminar on Railway Operations Modelling and Analysis (RailRome)*.

Bohlin, M., Flier, H., Maue, J., & Mihalák, M. (2011b). Track allocation in freight-train classification with mixed tracks. In A. Caprara, & S. Kontogiannis (Eds.), *11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems* (pp. 38–51). doi:10.4230/OASIcs.ATMOS.2011.38.

Bohlin, M., Gestrelius, S., Dahms, F., Mihalák, M., & Flier, H. (2016). Optimization methods for multistage freight train formation. *Transportation Science*, *50*, 823–840. doi:10.1287/trsc.2014.0580.

Bohlin, M., Hansmann, R. S., & Zimmermann, U. T. (2018). Optimization of railway freight shunting. In R. Borndörfer, T. Klug, L. Lamorgese, C. Mannino, M. Reuther, & T. Schlechte (Eds.), *Handbook of Optimization in the Railway Industry* (pp. 181–212). Springer volume 268 of *International Series in Operations Research & Management Science*. doi:10.1007/978-3-319-72153-8_9.

Borndörfer, R., Klug, T., Lamorgese, L., Mannino, C., Reuther, M., & Schlechte, T. (Eds.) (2018). *Handbook of Optimization in the Railway Industry* volume 268 of *International Series in Operations Research & Management Science*. Springer. doi:10.1007/978-3-319-72153-8.

Boysen, N., Emde, S., & Fliedner, M. (2016). The basic train makeup problem in shunting yards. *OR Spectrum*, *38*, 207–233. doi:10.1007/s00291-015-0412-0.

Boysen, N., Fliedner, M., Jaehn, F., & Pesch, E. (2012). Shunting yard operations: Theoretical aspects and applications. *European Journal of Operational Research*, *220*, 1–14. doi:10.1016/j.ejor.2012.01.043.

Bredström, D., & Rönnqvist, M. (2008). Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European Journal of Operational Research*, *191*, 19–31. doi:10.1016/j.ejor.2007.07.033.

Bruckmann, D. (2006). *Entwicklung einer Methode zur Abschätzung des containerisierbaren Aufkommens im Einzelwagenverkehr und Optimierung der Produktionsstruktur*. Ph.D. thesis Universität Duisburg-Essen Essen.

Büsing, C., & Maue, J. (2010). Robust algorithms for sorting railway cars. In M. de Berg, & U. Meyer (Eds.), *Algorithms - ESA 2010* (pp. 350–361). Springer, Berlin volume 6346 of *Lect. Notes Comput. Sci.*. doi:10.1007/978-3-642-15775-2_30.

Cacchiani, V., & Toth, P. (2012). Nominal and robust train timetabling problems. *European Journal of Operational Research*, *219*, 727–737. doi:10.1016/j.ejor.2011.11.003.

Caimi, G., Fischer, F., & Schlechte, T. (2018). Railway track allocation. In R. Borndörfer, T. Klug, L. Lamorgese, C. Mannino, M. Reuther, & T. Schlechte (Eds.), *Handbook of Optimization in the Railway Industry* (pp. 141–159). Springer volume 268 of *International Series in Operations Research & Management Science*. doi:10.1007/978-3-319-72153-8_7.

Christiaens, J., & Vanden Berghe, G. (2018). *Slack Induction by String Removals for Vehicle Routing Problems*. Technical Report KU Leuven.

Cicerone, S., D'Angelo, G., Di Stefano, G., & Frigioni, D. (2009a). Recoverable robustness for train shunting problems. *Algorithmic Operations Research*, *4*, 102–116. doi:10.1007/978-3-642-05465-5_2.

Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A., Schachtebeck, M., & Schöbel, A. (2009b). Recoverable robustness in shunting and timetabling. In R. K. Ahuja, R. H. Möhring, & C. D. Zaroliagis (Eds.), *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems* number 5868 in Lecture Notes in Computer Science. Berlin / Heidelberg: Springer. doi:10.1007/978-3-642-05465-5_2.

Clausen, U., & Voll, R. (2013). A comparison of North American and European railway systems. *European Transport Research Review*, *5*, 129–133. doi:10.1007/s12544-013-0090-4.

Cordeau, J.-F., Desaulniers, G., Lingaya, N., Soumis, F., & Desrosiers, J. (2001). Simultaneous locomotive and car assignment at VIA Rail Canada. *Transportation Research Part B*, *35*, 767–787. doi:10.1016/S0191-2615(00)00022-9.

Cordeau, J.-F., Laporte, G., Pasin, F., & Ropke, S. (2010). Scheduling technicians and tasks in a telecommunications company. *Journal of Scheduling*, *13*, 393–409. doi:10.1007/s10951-010-0188-7.

Cordeau, J.-F., Toth, P., & Vigo, D. (1998). A survey of optimization models for train routing and scheduling. *Transportation Science*, *32*, 380–404. doi:10.1287/trsc.32.4.380.

Courcoubetis, C., Vardi, M., Wolper, P., & Yannakakis, M. (1992). Memory-efficient algorithms for the verification of temporal properties. In *Formal Methods in System Design* (pp. 275–288). Kluwer Academic Publishers volume 1. doi:10.1007/BF00121128.

Crainic, T. G., Ferland, J.-A., & Rousseau, J.-M. (1984). A tactical planning model for rail freight transportation. *Transportation Science*, *18*, 165–184. doi:10.1287/trsc.18.2.165.

Crevier, B., Cordeau, J.-F., & Savard, G. (2012). Integrated operations planning and revenue management for rail freight transportation. *Transportation Research Part B: Methodological*, *46*, 100–119. doi:10.1016/j.trb.2011.09.002.

Daganzo, C. F., Dowling, R. G., & Hall, R. W. (1983). Railroad classification yard throughput: The case of multistage triangular sorting. *Transportation Research Part A*, *17A*, 95–106. doi:10.1016/0191-2607(83)90063-8.

Dahlhaus, E., Horak, P., Miller, M., & Ryan, J. F. (2000). The train marshalling problem. *Discrete Applied Mathematics*, *103*, 41–54. doi:10.1016/S0166-218X(99)00219-X.

Dejax, P. J., & Crainic, T. G. (1987). Survey paper - a review of empty flows and fleet management models in freight transportation. *Transportation Science*, *21*, 227–248. doi:10.1287/trsc.21.4.227.

Di Stefano, G., Maue, J., Modelski, M., Navarra, A., Nunkesser, M., & van den Broek, J. (2007). *Models for Rearranging Train Cars*. Technical Report ARRIVAL: Algorithms for Robust and online Railway optimization: Improving the Validity and reliAbility of Large scale systems.

Drexl, M. (2012). Synchronization in Vehicle Routing - A Survey of VRPs with Multiple Synchronization Constraints. *Transportation Science*, *46*, 297–316. doi:10.1287/trsc.1110.0400.

European Commission (2015). Study on Single Wagonload Traffic in Europe - challenges, prospects and policy options. *Final report*, (pp. 1–257).

Fisler, K., Fraer, R., Kamhi, G., Vardi, M. Y., & Yang, Z. (2001). Is there a best symbolic cycle-detection algorithm? In T. Margaria, & W. Yi (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems* (pp. 420–434). Springer volume 2031 of *Lecture Notes in Computer Science*. doi:10.1007/3-540-45319-9_29.

Fügenschuh, A., Homfeld, H., Johann, M., Schülldorf, H., & Stieber, A. (2018). Use of optimization tools for routing in rail freight transport. In R. Borndörfer, T. Klug, L. Lamorgese, C. Mannino, M. Reuther, & T. Schlechte (Eds.), *Handbook of Optimization in the Railway Industry* (pp. 161–179). Springer volume 268 of *International Series in Operations Research & Management Science*. doi:10.1007/978-3-319-72153-8_8.

Fügenschuh, A., Homfeld, H., & Schülldorf, H. (2015). Single-car routing in rail freight transport. *Transportation Science*, *49*, 130–148. doi:10.1287/trsc.2013.0486.

Gatto, M., Maue, J., Mihalák, M., & Widmayer, P. (2009). Shunting for dummies: An introductory algorithmic survey. *Robust and Online Large-Scale Optimization*, (pp. 310–337). doi:10.1007/978-3-642-05465-5_13.

Gestrelius, S., Aronsson, M., Joborn, M., & Bohlin, M. (2017). Towards a comprehensive model for track allocation and roll-time scheduling at marshalling yards. In *RailLille 2017 – 7th International Conference on Railway Operations Modelling and Analysis*. doi:10.1016/j.jrtpm.2017.06.002.

Grangier, P., Gendreau, M., Lehuédé, F., & Rousseau, L.-M. (2016). An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. *European Journal*

of Operational Research, *254*, 80–91. doi:10.1016/j.ejor.2016.03.040.

Gurobi Optimization, L. (2019). Gurobi optimizer reference manual. *http://www.gurobi.com*, .

Haghani, A. E. (1987). Rail freight transportation: A review of recent optimization models for train routing and empty car distribution. *Journal of Advances Transportation*, *21*, 147–172. doi:10.1002/atr.5670210205.

Haghani, A. E. (1989). Formulation and solution of a combined train routing and makeup, and empty car distribution model. *Transportation Research Part B*, *23B*, 433–452. doi:10.1016/0191-2615(89)90043-X.

Hansmann, R. S., & Zimmermann, U. T. (2008). Optimal sorting of rolling stock at hump yards. *Mathematics–Key Technology for the Future*, (pp. 189–203). doi:10.1007/978-3-540-77203-3_14.

He, S., Song, R., & Chaudhry, S. S. (2000). Fuzzy dispatching model and genetic algorithms for railyards operations. *European Journal of Operational Research*, *124*, 307–331. doi:10.1016/S0377-2217(99)00383-5.

He, S., Song, R., & Chaudhry, S. S. (2003). An integrated dispatching model for rail yards operations. *Computers & Operations Research*, *30*, 939–966. doi:10.1016/S0305-0548(02)00064-3.

Homfeld, H. (2012). *Consolidating Car Routes in Rail Freight Service by Discrete Optimization*. München: Verlag Dr. Hut.

IBM ILOG CPLEX (2015). *IBM ILOG CPLEX Optimization Studio CPLEX User's Manual*. Technical Report 6 IBM. URL: https://www.ibm.com/support/knowledgecenter/SSSA5P_12.6.3/ilog.odms.studio.help/pdf/usrcplex.pdf.

Jacob, R., Márton, P., Maue, J., & Nunkesser, M. (2011). Multistage methods for freight train classification. *Networks*, *57*. doi:10.1002/net.20385.

Jaehn, F., Rieder, J., & Wiehl, A. (2015). Single-stage shunting minimizing weighted departure times. *Omega*, *52*, 133–141. doi:10.1016/j.omega.2014.11.001.

Keaton, M. H. (1992). Designing railroad operating plans: A dual adjustment method for implementing lagrangian relaxation. *Transportation Science*, *26*, 263–279. doi:10.1287/trsc.26.4.263.

Kraft, E. R. (2000). A hump sequence algorithm for real time management of train connection reliability. *Journal of the Transportation Research Forum*, *39*, 95–115.

Kraft, E. R. (2002a). Priority-based classification for improving connection reliability in railroad yards: Part I. Integration with car scheduling. *Journal of the Transportation Research Forum*, *56*, 93–105.

Kraft, E. R. (2002b). Priority-based classification for improving connection reliability in railroad yards: Part II. Dynamic block to track assignment. *Journal of the Transportation Research Forum*, *56*, 107–119.

Kroon, L. G., Lentink, R. M., & Schrijver, A. (2008). Shunting of passenger train units: An integrated approach. *Transportation Science*, *42*, 436–449. doi:10.1287/trsc.1080.0243.

Lamorgese, L., & Mannino, C. (2015). An exact decomposition approach for the real-time train dispatching problem. *Operations Research*, *63*, 48–64. doi:10.1287/opre.2014.1327.

Li, H., Song, R., Jin, M., & He, S. (2014). Optimization of railcar connection plan in a classification yard. In *Transportation Research Board 93rd Annual Meeting*.

Lübbecke, M., & Zimmermann, U. T. (2005). Shunting minimal rail car allocation. *Computational Optimization and Applications*, *31*, 295–308. doi:10.1007/s10589-005-3229-y.

Lusby, R. M., Larsen, J., Ehrgott, M., & Ryan, D. (2011). Railway track allocation: models and methods. *OR Spectrum*, *33*, 843–883. doi:10.1007/s00291-009-0189-0.

Mancera, A., Bruckmann, D., & Weidmann, U. (2015). Single wagonload production schemes improvements using gütersim (agent-based simulation tool). *Transportation Research Procedia*, *10*, 615–624. doi:10.1016/j.trpro.2015.09.015.

Marinov, M., Woroniuk, C., & Zunder, T. H. (2012). Recent developments with single wagon load services, policy and practice in europe. In *Proceedings of the Federated Conference on Computer Science and Information Systems* (pp. 1097–1104). Wrocław.

Márton, P., Maue, J., & Nunkesser, M. (2009). An improved train classification procedure for the hump yard lausanne triage. In J. Clausen, & G. Di Stefano (Eds.), *ATMOS 2009* (pp. 1–15). doi:10.4230/OASIcs.ATMOS.2009.2142.

Masson, R., Lehuédé, F., & Péton, O. (2013). Efficient feasibility testing for request insertion in the pickup and delivery problem with transfers. *Operations Research Letters*, *41*, 211–215. doi:10.1016/j.orl.2013.01.007.

Maue, J., & Nunkesser, M. (2009). *Evaluation of computational methods for freight train classification schedules*. Technical Report ARRIVAL Project.

Milenkovic, M., & Bojovic, N. (2019). *Optimization Models for Rail Car Fleet Management*. Amsterdam: Elsevier. doi:10.1016/C2017-0-03011-2.

Mortimer, P., & Islam, D. M. Z. (2014). A comparison of North American and European railway systems – a critique and riposte. *European Transport Research Review*, *6*, 503–510. doi:10.1007/s12544-014-0148-y.

Petersen, E. R. (1977). Railyard Modeling: Part I. Prediction of Put-Through Time. *Transportation Science*, *11*, 37–49. doi:10.1287/trsc.11.1.37.

Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, *34*, 2403–2435. doi:10.1016/j.cor.2005.09.012.

Pisinger, D., & Ropke, S. (2010). Large neighborhood search. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* International Series in Operations Research and Management Science (pp. 399–419). New York: Springer. doi:10.1007/978-1-4419-1665-5_13.

Piu, F., & Speranza, M. G. (2014). The locomotive assignment problem: a survey on optimization models. *International Transactions in Operational Research*, *21*, 327–352. doi:10.1111/itor.12062.

Posner III, H. (2008). Rail Freight in the USA: Lessons for Continental Europe. In *CER Essay Series*.

Potvin, J.-Y., & Rousseau, J.-M. (1993). A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, *66*, 331–340. doi:10.1016/0377-2217(93)90221-8.

Preis, H., Frank, S., Bäcker, S., & König, R. (2018). Optimizing production schedules in classification yards. *Proceedings of the 7. Transport Research Arena Vienna*, . doi:10.5281/zenodo.1483764.

Raut, S., Sinha, S. K., Khadilkar, H., & Salsingikar, S. (2019). A rolling horizon optimisation model for consolidated hump yard operational planning. *Journal of Rail Transport Planning & Management*, *9*, 3–21. doi:10.1016/j.jrtpm.2018.09.002.

Ropke, S., & Pisinger, D. (2006a). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, *40*, 455–472. doi:10.1287/trsc.1050.0135.

Ropke, S., & Pisinger, D. (2006b). A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, *171*, 750–775. doi:10.1016/j.ejor.2004.09.004.

Savelsbergh, M. W. P. (1991). The vehicle routing problem with time windows: minimizing route duration. *Memorandum COSOR*, *9103*. doi:10.1287/ijoc.4.2.146.

Scheffler, M., Neufeld, J. S., & Hölscher, M. (2020). An MIP-based heuristic solution approach for the locomotive assignment problem focussing on (dis-)connection processes. *Transportation Research Part B: Methodological*, *139*, 64–80. doi:10.1016/j.trb.2020.05.020.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., & Dueck, G. (2000). Record breaking optimization results

using the ruin and recreate principle. *Journal of Computational Physics*, *159*, 139–171. doi:10.1006/jcph.1999.6413.

Shaw, P. (1997). *A new local search algorithm providing high quality solutions to vehicle routing problems*. Technical Report Department of Computer Science, University of Strathclyde.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher, & J.-F. Puget (Eds.), *Principles and Practice of Constraint Programming - CP98* (pp. 417–431). 4th International Conference, CP 98 Springer. doi:10.1007/3-540-49481-2_30.

Turnquist, M. A., & Daskin, M. S. (1982). Queuing models of classification and connection delay in railyards. *Transportation Science*, *16*, 207–230. doi:10.1287/trsc.16.2.207.

Vidal, R. V. V. (1993). *Applied Simulated Annealing*. Lecture Notes in Economics and Mathematical Systems. Berlin / Heidelberg: Springer-Verlag. doi:10.1007/978-3-642-46787-5.

Xiao, J., Pachl, J., Lin, B., & Wang, J. (2018). Solving the block-to-train assignment problem using the heuristic approach based on the genetic algorithm and tabu search. *Transportation Research Part B*, *108*, 148–171. doi:10.1016/j.trb.2017.12.014.

Yagar, S., Saccomanno, F. F., & Shi, Q. (1983). An efficient sequencing model for humping in a rail yard. *Transportation Research Part A*, *17A*, 251–262. doi:10.1016/0191-2607(83)90089-4.

Zhang, Y., Song, R., He, S., Li, H., & Guo, X. (2018). Optimization of classification track assignment considering block sequence at train marshaling yard. *Journal of Advances Transportation*, (pp. 1–11). doi:10.1155/2018/3802032.