Personnel Scheduling on Railway Yards

Roel van den Broek
Department of Computer Science, Utrecht University, The Netherlands
r.w.vandenbroek@uu.nl

Han Hoogeveen
Department of Computer Science, Utrecht University, The Netherlands
j.a.hoogeveen@uu.nl

Marjan van den Akker
Department of Computer Science, Utrecht University, The Netherlands
j.m.vandenakker@uu.nl

Abstract

In this paper we consider the integration of the personnel scheduling into planning railway yards. This involves an extension of the Train Unit Shunting Problem, in which a conflict-free schedule of all activities at the yard has to be constructed. As the yards often consist of several kilometers of railway track, the main challenge in finding efficient staff schedules arises from the potentially large walking distances between activities.

We present two efficient heuristics for staff assignment. These methods are integrated into a local search framework to find feasible solutions to the Train Unit Shunting Problem with staff requirements. To the best of our knowledge, this is the first algorithm to solve the complete version of this problem. Additionally, we propose a dynamic programming method to assign staff members as passengers to train movements to reduce their walking time. Furthermore, we describe several ILP-based approaches to find a feasible solution of the staff assignment problem with maximum robustness, which solution we use to evaluate the quality of the solutions produced by the heuristics.

On a set of 300 instances of the train unit shunting problem with staff scheduling on a real-world railway yard, the best-performing heuristic integrated into the local search approach solves 97% of the instances within three minutes on average.

2012 ACM Subject Classification Applied computing → Transportation; Computing methodologies → Planning for deterministic actions

Keywords and phrases Staff Scheduling, Train Shunting, Partial Order Schedule

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.12

1 Introduction

Passenger railway operators use only a subset of the available trains during off-peak hours. Railway yards are used to store the surplus of rolling stock and often provide cleaning and maintenance services for the parked trains.

To ensure that the yards are operating efficiently and that all trains leave the yard in the correct composition and at their scheduled departure time, the Dutch passenger railway operator *NS* requires that a *shunting plan* is created in advance. A shunting plan describes the activities, such as coupling and decoupling train units, service tasks and train movements, that need to be performed together with their time intervals and locations.

The activities in the shunting plan have to be performed by skilled staff members. Previous work on algorithms for constructing feasible shunting plans assumes that sufficient staff is available on the yard to complete all activities as planned. However, in practice personnel is a scarce resource, and hence their availability has a large impact on the feasibility of a shunting plan. Therefore, in this paper we consider the integration of the staff scheduling into a yard planning approach.

Due to the sheer size of a shunting yard, the number of tasks that an employee can perform is severely limited by the walking distances between the locations of consecutive tasks. For example, if a driver is assigned to two train movements, and the destination of the first movement is far away from the start of the second movement, then the walking time between these two locations can easily be more than the total driving time of the two train movements combined. Even in the case that a train has two consecutive movements in opposite directions, and the driver continues to operate the same train, then the driver still has to walk from one end of the train to the other to ensure that he or she is looking in the driving direction.

As service and movement tasks take place at many locations on the yard, walking is often unavoidable. However, ordering the tasks properly and carefully dividing the tasks over the available employees can significantly reduce the walking time, which is essential in constructing shunting plans in which all service and movement tasks can be performed on time (while keeping the personnel satisfied).

In this paper we first give an overview of recent literature related to shunting and staff rostering problems in Section 2, and continue with a formal introduction to the staff assignment problem in Section 3. We then propose two solution methods for the staff assignment problem, a list scheduling procedure and a decomposition approach, in Section 4. In this section we further present several approaches to solve the staff scheduling problem to optimality. We compare the two heuristic methods in Section 5 in an experimental study and formulate some concluding remarks in Section 6.

2 Literature Overview

The train unit shunting problem (TUSP) was first introduced by [3] and consists of parking passenger trains that arrive at a station or a railway yard on the available tracks and assigning these trains to the scheduled departures in the timetable. The problem formulation was later extended by [6] to include the paths taken by the trains over the yard.

To construct feasible solution to the TUSP, [5] decompose the problem into two sub-problems that are solved sequentially with mixed integer programming techniques. In the first sub-problem they match the arriving trains to the departure compositions and assign the trains to the parking tracks. The second sub-problem consists of assigning paths and start times to the train movements resulting from the solution of the first sub-problem.

The extension of the train unit shunting problem with service tasks is studied in [9]. These service tasks, such as maintenance checks and cleaning, can be performed at facilities located on the railway yard and have to be completed before the train departs from the yard. We presented a local search method that solves the train unit shunting problem with service tasks by iteratively modifying the current solution to resolve conflicts.

In this paper we address the integration of staff scheduling into the train unit shunting problem. For a broad overview of staff scheduling problems and solution methods we refer to [7]. A prominent feature of scheduling personnel at shunting yards is the (often large) walking distance between consecutive tasks. [4] introduce the closely related service technician routing and scheduling problem (STRSP), in which staff members have to be assigned to tasks that have time windows, precedence constraints and geographic locations. The authors use an adaptive large neighborhoods search to find solutions for the STRSP.

A survey on more general personnel scheduling and routing problems is given in [1]. They provide an overview of common constraints, application domains and solution methods of scheduling and routing problems. Furthermore, they performed numerical experiments to measure the running times of several (mixed) integer programming formulations of these problems.

In contrast to the problems described by [1], the staff scheduling problem at railway yards contains tasks, specifically train movements, that start and end at different geographical locations. Furthermore, we do not consider the staff scheduling problem as an isolated problem. Instead, the staff scheduling is only one component of the shunting process, and the computational complexity arises mainly from the interactions of the strongly intertwined sub-problems. Therefore, the primary contribution of this paper is that we develop solution methods for the staff scheduling problem which can be embedded in a larger framework to construct plans for the complete shunting problem. The integrated solution approach that we propose in this paper is, to our knowledge, the first method capable of solving the shunting problem with service activities and staff scheduling on real-world instances.

3 Problem Description

Shunting yards are a collection of tracks, connected by switches, where the rolling stock of passenger railway operators are stored. Modern trains are typically *electrical multiple unit* trains, which are self-propelled, permanently coupled carriages. We refer to a single electrical multiple unit as a *train unit*. These train units can be coupled to transport more passengers. A *train* is a group of one or more train units that are coupled. The train units are classified by their *train type*.

Trains arrive at and depart from the railway yard according to the timetable, which specifies for each arrival and departure the scheduled time and train types of which the train is composed.

The shunting problem at railway yards consists of six components:

1. matching incoming train units to positions in outgoing trains;
2. (de-)coupling trains to form the correct train compositions for departure;
3. scheduling all required service activities such that they are completed before the trains depart;
4. parking the trains on the yard;
5. finding conflict-free paths for all train movements;
6. assigning staff to all the train activities.

Earlier work focused primarily on the first five problem components. An in-depth description of these components can be found in [8, 9].

Any solution to the problem – a *shunting plan* – can be represented by a set of activities A, a *partial order schedule* $\mathcal{POS}$ imposing precedence constraints on A, and a *scheduling policy* that assigns start times to the activities in A based on the $\mathcal{POS}$. A solution is feasible if

- A contains a valid sequence of activities for each train unit with respect to their required service activities and the infrastructure of the yard;
- the split and combine activities in A induce a feasible matching;
- the constraints in the $\mathcal{POS}$ ensure that no resource and routing conflicts occur;
- the start times assigned to the activities by the scheduling policy satisfy both the precedence constraints in the $\mathcal{POS}$ and the timetable constraints.

In this paper we focus on the sixth problem component, the staff scheduling sub-problem. That is, we assume that the activities in A require one or more skilled staff members. Since the staff assignment might impose additional precedence constraints on the activities, we now consider the problem of assigning both staff members and start times to each of the activities, given the partial order schedule obtained from solving the first five shunting sub-problems.

The staff members are grouped by their skill set, i.e., the activity types that they are qualified to perform. Each activity $a \in A$ has

- a train t_a,
- a duration d_a,
- a track τ_a^{init} on which the activity is initiated,
- a track τ_a^{final} where the activity ends,
- a staffing requirement of r_a^T staff members of type T.

Note that $\tau_a^{init} \neq \tau_a^{final}$ if and only if the activity is a train movement. We have to assign to each activity a sufficient staff members, as well as a feasible start time st_a and completion time $ct_a = st_a + d_a$.

For every available staff member in the planning horizon, we are given their skill set type $T \in \mathcal{T}$. The skill sets are typically disjunct sets, i.e., train drivers may only move trains, cleaners clean the trains and mechanics are limited to repairing and inspecting the trains. Furthermore, for each pair (τ_i, τ_j) of tracks we have the *walking duration* ω_{τ_i,τ_j}, which is the time required for a staff member to walk from track τ_i to track τ_j. Note that a more detailed model of the walking durations can be used if the exact locations of the train activities on the tracks (e.g. in meters) are known.

The staff assignment sub-problem is now to assign staff members and a start time to each of the activities in the shunting plan such that

- sufficient staff members with the correct skills are assigned to each activity;
- all activities of a staff member are scheduled without overlap;
- the start times of the activities satisfy the precedence constraints in the $\mathcal{POS}$;
- the time between activities in the schedule of a staff member is at least equal to the necessary walking duration.

For train drivers there are additional constraints associated with the scheduling of train movements. If a train driver is assigned to two consecutive movements of a train t in opposite directions, then the driver has to walk to the driver's compartment at the other end of the train to reverse the movement direction of the train. This *reversal* occurs after the completion of the first movement and before the start of the second movement. The duration of the reversal, which we denote by d^t_{reversal}, depends linearly on the length of the train t.

Although a train movement only requires a single train driver, additional train drivers are allowed to be in the train during the movement. This enables the drivers to move more efficiently over the yard, but it creates additional dependencies between the schedules of the drivers. Making a detour or stopping for (dis)-embarking is not allowed.

As an example, suppose that we have a sequence of five consecutive train movements, $a_1, \ldots, a_5$, in our shunting plan and two drivers x and y available at the yard. Movements a_1, a_2, a_4 and a_5 are from track τ_1 to τ_2, and movement a_3 is in the opposite direction, starting at τ_2 and ending at τ_1. A possible assignment of the drivers to the movements is that x performs a_1, a_3 and a_5 and y operates movements a_2 and a_4. Then driver x moves between tracks τ_1 and τ_2 by train, but driver y needs to get from τ_2 to τ_1 to perform a_4. As train movement a_3 is going from τ_2 to τ_1 anyway, driver y can save some walking time by joining x on a_3.

The objective of the train unit shunting problem – and therefore the staff scheduling sub-problem – is primarily to find a feasible shunting plan. However, in practice the solutions have to be robust to small disturbances as well, such that a small delay earlier in the execution will not make the plan infeasible. Therefore, we include the maximization of total free slack in the objective as a measure of robustness of the solutions.

4 Solution Methods

In [9] we present a local search framework that searches for feasible solutions to the first five components of the shunting problem. To find shunting and service plans that satisfy the additional staff scheduling constraints introduced in the previous section, we propose two methods that can be integrated as sub-routines within the local search framework. These methods use the information in the partial order schedules generated by the local search to assign the train activities to their required resources, which includes the personnel.

The first method is a greedy *list scheduling policy*. Here we transform the $\mathcal{POS}$ in an ordered list of activities and assign to each activity the earliest possible starting time given the starting times of the earlier activities in the list and the availability of the staff.

In the first method we do not attempt to maximize the slack or minimize the walking distances of the staff; we simply pick the first available person for each activity. In the second method, we *decompose* the problem into the sub-problems of assigning activities to the staff, with weights based on the walking distances and the slack between activities, and then compute start times using the first method. If this does not result in a feasible solution, then we re-assign the activities to the staff, etc.

To evaluate the quality of the solutions produced by these two heuristics, we compare them to optimal staff assignments for the given partial order schedule with the objective of maximizing total slack. To construct these exact solutions we present several mixed integer linear programs.

In the remainder of this section we will describe the proposed methods in more detail.

4.1 List Scheduling Policy

In the list scheduling policy we use an ordered list L of the activities in the shunting plan. L is a linearization of the partial order schedule, i.e., a total ordering of the activities. The list L defines the order in which we will evaluate the activities to assign start times to them. See Section 4.3 for a description of the construction and modification of the priority list.

When we evaluate activity a_i in $L = (a_1, \dots, a_n)$, activities a_j with $j < i$ have already been assigned a start time and staff member by the procedure. To compute the start time of a_i, we have to determine the availability of

- the train t_{a_i};
- infrastructure: cleaning platforms or train movement tracks and switches;
- staff: train drivers, mechanics or cleaners.

In the list scheduling approach we consider all the above as resources on which the activity has to be scheduled.

For each resource R required by a_i, we compute the set of feasible time windows T_R in which a_i can start on R given the duration of a_i and the activities a_j with $j < i$ that have already been scheduled on R. For resources with a capacity larger than one, e.g. multiple train drivers or cleaning crews, the set of feasible start time windows consists of all time windows in which there is sufficient capacity available of the resource to process the activity.

Once we have the set of feasible start times of each resource, we compute the start time of a_i as the earliest time that is feasible for all resources. Note that list L only indicates the priority of the activities, and not their actual order. Therefore, a_i can start before activity a_j with $j < i$ if the required resources allow this. We assign staff to activity a_i by retracing the staff members that contributed to the feasible time window at the computed start time of a_i.

Recall that the walking time of a train driver can be decreased by riding along with another train movement. To add this feature to the list scheduling policy, we take the scheduled train movements into account when computing the minimum time lag between two train movements assigned to the same driver. Let a_i be the movement activity that we are currently scheduling in our list scheduling policy. To determine when a train driver can perform a_i after the scheduled train movement $a_{i'}$, with $i' < i$, we compute the minimum time lag between $a_{i'}$ and a_i using a dynamic programming approach. Let $M^i_{i'} = \{m_1, \dots, m_k\}$ be the set of movement activities m_j with $j < i$ that start after $a_{i'}$, ordered by their start time. Then $M^i_{i'}$ is the set of train movements that the driver can board to get faster from $a_{i'}$ to a_i. Define D^j_τ as the earliest time that the driver can reach track τ with the possibility of traveling with trips from $\{m_1, \dots, m_j\}$. Similarly, let D^0_τ be the time that the driver reaches track τ after completing $a_{i'}$ and walking from the destination of $a_{i'}$ to τ, i.e. $D^0_\tau = ct_{a_{i'}} + \omega_{\tau^{final}_{a_{i'}},\tau}$. Then we compute D^j_τ for all $1 \leq j \leq k$ and all tracks τ as

$$D^j_\tau = \begin{cases} D^{j-1}_\tau & \text{if } D^{j-1}_{\tau^{init}_{m_j}} > st_j \\ \min\left\{D^{j-1}_\tau, ct_j + \omega_{\tau^{final}_{m_j},\tau}\right\} & \text{otherwise .} \end{cases} \tag{1}$$

That is, D^j_τ is the minimum of D^{j-1}_τ – the case in which the driver does not travel with movement m_j – and the completion time of m_j plus the walking time from the destination $\tau^{final}_{m_j}$ of m_j to τ. The latter is only possible if the driver can reach the initial location of m_j before m_j departs.

The earliest time that the driver can start movement activity a_i after $a_{i'}$ is then $D^k_{\tau^{init}_{a_i}}$. We take the minimum over all $i' < i$ to compute the earliest time that activity a_i can start.

4.2 Decomposition Heuristic

With the decomposition heuristic, we first solve the problem of assigning activities to personnel. The assignment gives us the schedules of the individual staff members, which we will combine with the precedence constraints from the partial order schedule to compute the start times of the activities in the second step.

We solve the staff assignment problem for each type of personnel T (train drivers, mechanics and cleaners) separately. Let $A_T = \{a_1, \dots, a_n\}$ be the set of activities that require staff members of type T, and define k_T as the number of staff members available. We construct the following directed graph $G = (V, A)$. We let each activity a_j correspond to a vertex v_j in V, and we add the arc (v_i, v_j) if it is possible for a staff member to carry out activities a_i and a_j consecutively in that order. Since the $\mathcal{POS}$ does not correspond to a full order, G may contain cycles. To avoid this, we remove all arcs in A that do not comply with the list L defined in the previous sub-section. Remark that this ordering only matters if the tasks a_i and a_j are assigned to the same staff member.

A schedule for a staff member now corresponds to a path in the graph G. Given the weights of the arcs (v_i, v_j), which we define later, we can then solve the assignment problem as a min-cost max-flow problem with a minimum flow of 1 through each vertex and an upper bound of k_T on the size of the flow. We follow the approach of [2] to solve this problem as a weighted matching problem in a bipartite graph.

To find a maximum matching that is likely to satisfy the departure time constraints, we assign weights to the arcs in the bipartite graph. For each activity $a_i \in A_T$, we compute its earliest completion time ect_i and latest start time lst_i in the original partial order schedule. Then, for every arc (v_i, v_j) we set the weight of the arc to

$$w_{i,j} = lst_j - ect_i - mtl_{i,j}, \tag{2}$$

where $mtl_{i,j}$ is the minimum time lag due to walking between a_i and a_j. With these weights, matchings in which the staff has sufficient slack time between their scheduled activities are preferred. Note that we cannot guarantee that a maximum weighted matching causes no departure delays, as the arc weights are only pair-wise indications of the feasibility of performing two activities consecutively. From the solutions of the staff assignment problem we obtain a set of minimum time lag constraints C between the activities that extend the precedence relations in the partial order schedule $\mathcal{POS}$. We can then find the starting times of the activities using the dynamic programming approach from Section 4.1 in combination with the priority list L.

If the resulting solution is not feasible with respect to the train departure times, then we update the staff assignment problem by updating the weights of the arcs. We select the matching variables that have their corresponding precedence relation on a critical path causing the delays, and reduce their weight by the total amount of departure delay resulting from the critical path. Then, we solve the two sub-problems again. This procedure is repeated until we either find a feasible solution, fail to generate a new solution, or reach the predefined maximum number of iterations.

4.3 Embedding in the Local Search

The two staff assignment heuristics are intended to extend a partial solution of the first five problem components defined in Section 3 – generated by the local search – to a full shunting plan including staff schedules. In addition to the $\mathcal{POS}$, the input of the two methods consists of a priority list L of the activities as well. Since the order of activities in the priority list affects the solutions produced by the two staff assignment algorithms, we have to allow the local search to modify the priority of the activities.

As described in [9], the local search method has several neighborhoods that change the order of the activities in the partial order schedule. When the local search modifies the ordering in the $\mathcal{POS}$, we update the priority list such that it remains a linearization of the partial ordering. Additionally, we introduce two new neighborhoods that change the ordering of L without affecting the $\mathcal{POS}$ by **shifting** and **swapping** activities, respectively.

4.4 Exact Formulations

In addition to the two heuristic approaches, we propose three exact models based on mathematical programming to construct the staff assignment of the train drivers based on the $\mathcal{POS}$ and the priority list L. The first method is a mixed integer linear program formulation and the other two methods are branch-and-price algorithms. Although the exact methods will be computationally too intensive to be implemented within the local search framework, it allows us to evaluate the quality of the solutions obtained with the heuristics by comparing them to optimal staff assignments.

Our primary goal is to find a feasible solution. Since the shunting yard is a very dynamic environment with many possible disturbances, we strive for robust solutions. We use the *free slack* times – the maximum amount of time an activity can be delayed without affecting other activities – as a measure for the robustness; we denote this by s_i and try to maximize the sum in our objective function. Instead of just maximizing the sum, we can also maximize a non-increasing piece-wise linear function of the slacks.

In the MIP model, we decide on the completion time $c_i \geq 0$ and the slack $s_i \geq 0$ of each activity $a_i \in L$. Furthermore, we have to assign the activities to staff members. We model the assignment as a flow problem in which the flow through the activities defines the staff schedules. Let a_0 and a_{n+1} be dummy activities representing the start and end of the shunting plan, respectively. For each pair of activities (a_i, a_j) with $0 \leq i < j \leq n+1$, we define the decision variables

$$x_{i,j} = \begin{cases} 1 & \text{if } a_i \text{ directly precedes } a_j \text{ in the schedule of a staff member,} \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

By assigning to each $a_i \in L$ a single predecessor and successor we construct the sequences of the activities in the staff schedules. Each schedule starts with a_0 and ends with a_{n+1}.

To formulate the staff assignment problem as a mixed integer linear program, we denote the data following from the $\mathcal{POS}$ and the staff schedules as

$$\begin{aligned}
ect_i &= \text{earliest completion time of activity } a_i \text{ in } \mathcal{POS} \\
lct_i &= \text{latest completion time of activity } a_i \text{ in } \mathcal{POS} \\
d_{ij} &= \begin{cases} \text{minimum time between } c_i \text{ and } c_j & \text{if } a_i \prec a_j \in \mathcal{POS} \\ \text{duration } d_{a_j} \text{ of activity } a_j & \text{otherwise} \end{cases} \\
D_{ij} &= \begin{cases} \text{minimum time between } c_i \text{ and } c_j & \text{if } a_i \prec a_j \in \mathcal{POS} \\ ect_j - lct_i & \text{otherwise} \end{cases} \\
\omega_{ij} &= \text{minimum walking time between } c_i \text{ and } c_j \\
m &= \text{number of available train drivers}
\end{aligned}$$

The problem of finding a staff assignment that maximizes the slack of the activities can then be formulated as

$$\max \sum_i s_i \qquad \text{s.t.} \tag{4}$$

$$c_j - c_i - s_i - Q_{ij} x_{i,j} \geq D_{ij} \qquad \forall i, j : 1 \leq i < j \leq n \tag{5}$$

$$c_i + s_i \leq lct_i \qquad \forall i \tag{6}$$

$$\sum_{j<i} x_{j,i} = 1 \qquad \forall i \in \{1, \dots, n\} \tag{7}$$

$$\sum_{j>i} x_{i,j} = 1 \qquad \forall i \in \{1, \dots, n\} \tag{8}$$

$$\sum_{j>0} x_{0,j} \leq m \tag{9}$$

$$x_{i,j} \in \{0,1\}, c_i \geq ect_i, s_i \geq 0 \tag{10}$$

where Q_{ij} is the additional time lag due to driver constraints,

$$Q_{ij} = \max \{\omega_{ij} - D_{ij}, 0\}.$$

In this model, the minimum time lag between activities is enforced by (5). The driver walking time Q_{ij} is included if and only if a_i is the direct predecessor of a_j in some staff schedule. Inequalities (6) provide the upper bounds on the completion times including slack of the activities. Each activity $a_i \in L$ is included in a single staff schedule due to the in- and outflow constraints (7) and (8). Constraint (9) models the maximum number of staff members available.

Our branch-and-price approaches are based on a covering formulation: we select for each staff member a schedule. The main difference between the two methods is whether the completion times of the activities are included in the schedules. In the *sequence model* the pricing problem determines only the sequence of activities in the individual staff schedules;

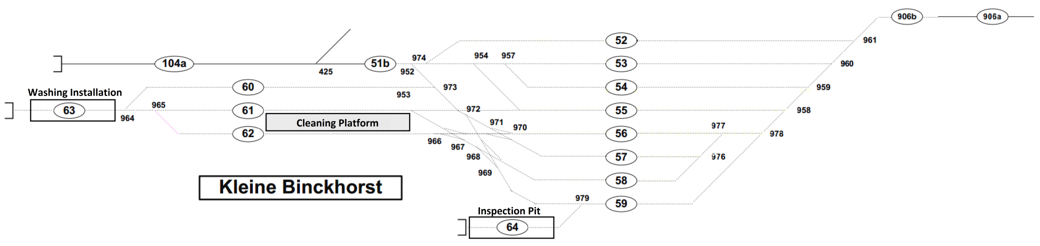

Figure 1 The "*Kleine Binckhorst*" shunting yard is situated near The Hague Central Station and is operated by NS.

the times at which these are done are decided by the master problem. In contrast, the completion time of activities in a staff schedule in the *timestamp model* are given by the pricing problem. Detailed descriptions of the master and pricing problems of the two methods are given in Appendix A.

Preliminary experiments have shown that the *sequence model* performs significantly better than the *timestamp model*. The former is able to solve our test instances in half an hour, whereas the latter takes several hours to compute an optimal staff assignment. The main bottleneck in the *timestamp model* appears to be the schedule generation, as the pricing problem takes far more time to solve when completion times have to be assigned to the activities in a schedule. However, the preliminary experiments also show that neither of the branch-and-price algorithms are competitive with the direct MIP formulation of the personnel rostering problem on our test instances. The MIP model is able to solve most instances within a few minutes, and, hence, we only use the mixed integer linear program in the remainder of our computational experiments. For the sake of completeness we have included the branch-and-price methods in the appendix.

5 Experimental Setup and Results

To compare the two solution methods, we will evaluate their performance on a set of instances generated for a real-world shunting yard. The *"Kleine Binkchorst (KBH)"*, shown in Figure 1, is a shunting yard of the NS near the central station of The Hague. For this location, we have generated 300 instances in which 13 to 15 train units arrive during the evening and have to depart the next morning. The arrival and departure times are sampled from an empirical distribution. All trains have to be cleaned internally.

All 300 instances are solvable by the local search algorithm described in [9] when we exclude the driver assignment component of the shunting problem. On average these solutions contain 72 train movements that have to be operated by a train driver.

To model the staff assignment problem, we included three train drivers that will be available during the entire planning horizon. The walking distances are provided by NS based on real-world data and range from 2 to 20 minutes, depending on the physical location of the tracks.

With the exact MIP model of the staff assignment problem, we have first determined whether feasible staff schedules exist for the solutions without drivers. In only 23 of the 300 cases the solver was able to construct feasible staff assignments, all other instances were infeasible. On average three activities remained unassigned in the infeasible instances, and the average computation time over all instances was 62 seconds. These experiments show that the staff assignment problem is not easily solvable as a post-processing step once a feasible solution to the other shunting sub-problems has been found, which suggests that an integrated solution method might perform better.

Table 1 The results for 300 instances of the *Kleine Binckhorst* yard in the Netherlands. The average computation time of feasible solutions produced by the heuristics is listed in seconds. The slack is denoted in minutes and is averaged over the feasible solutions as well.

Method	Solved Instances	Computation Time	Heuristic Slack	Optimal Slack
Complete LSP	276	54	403	557
Complete DH	196	414	476	579
Partial LSP	286	116	395	543
Partial DH	195	245	469	565

We tested four configurations of the local search on the 300 instances extended with the staff requirements. In the first two configurations we call the staff assignment sub-routine (either the list scheduling policy LSP or the decomposition heuristic DH) for every solution that the local search explores. We will refer to these two configurations as the *complete* LSP and the *complete* DH approaches.

In the two other configurations, we first search for a feasible solution without any staff assignment, similar to the baseline case where we did not include the train drivers in the instances. Once a feasible solution without personnel has been found, we run either one of the staff assignment heuristics. If the resulting solution contains delays, we continue the local search algorithm with the staff assignment sub-routine. These two configurations are listed in Table 1 as the *partial* LSP and *partial* DH methods. On each instance we ran the four local search variants until a feasible solution was found, or the maximum computation time of 1800 seconds was reached. In each iteration of these runs we used the basic walking durations and not the more sophisticated dynamic programming approach in Equation (1).

In Table 1 we list the average slack of the feasible solutions produced by the heuristics. To evaluate the quality of these solutions, we have computed the optimal staff assignments of the feasible partial ordering schedules using our MIP formulation. The optimal slack can be found in the last column of the table.

Table 1 shows the computational results of our experiments. All instances with train drivers have been solved successfully by at least one of the local search configurations. The average time required by the MIP to construct the optimal staff assignments is close to one minute regardless of the methods used to find the initial solutions.

The list scheduling policy outperforms the decomposition heuristic for both the complete and partial local search variants. The decomposition heuristic fails to find feasible solutions for one-third of the instances, whereas the list scheduling algorithm solves most of the instances relatively fast. This might indicate that the decomposition approach is not able to update the edge weights properly to converge to a good schedule for the train drivers.

Although fewer instances are solved by the decomposition heuristic, the solutions it produces have almost 20% more slack than the schedules constructed by the list scheduling policy. However, the exact solutions obtained with the MIP model are significantly more robust, and the computation times of the decomposition heuristic variants are higher than the average computation time of LSP plus the MIP. The longer computation time of the decomposition approach is most likely due to constructing and solving the matching problem in every iteration.

The differences between starting directly with the staff assignment or first searching for a feasible solution without staff are much smaller. In the case of the LSP approach, starting from a feasible solution without staff increases the number of instances that can be solved at the cost of doubling the computation time. This suggests that adapting the solution without staff to the staffing constraints might require either many small modifications, or several high-cost intermediate solutions that are unattractive for the local search to explore.

Of the four local search configurations, scheduling the staff with the list scheduling policy for every candidate solution shows the best performance. The computation time of this configuration is close to the local search without staff assignment, and 95% of the instances are solved. When we allow in each iteration the drivers to travel as passengers with planned train movements using the dynamic program in Equation (1), then the best configuration is capable of solving 292 instances. Although with the dynamic program we are able to solve 97% of the instances, it comes at the cost of a moderate increase in computation time to, on average, 162 seconds. The average slack in these solutions drops slightly to 382 minutes. As our MIP model does not support drivers as passengers, we have not not computed the optimal slack of these instances.

6 Conclusion

In this paper, we have studied the extension of the train unit shunting problem with staffing constraints. We proposed two methods that construct staff schedules for a given partial ordering of the activities on the shunting yard. The first method implements a list scheduling policy to distribute the activities of the available personnel, whereas the second approach decomposes the problem into a staff assignment and a time assignment sub-problem that are solved iteratively. These methods can be used in conjunction with the local search presented in [9] to find feasible shunting plans that fully integrate all components of the planning problem at the railway yards. Additionally, we presented a MIP model to compute the staff assignment that maximizes the total free slack in the complete shunting plan given the partial order schedule.

We studied the performance of the solution methods on a set of 300 realistic instances of the "Kleine Binckhorst" shunting yard. The experiments show that the list scheduling approach outperforms the decomposition heuristic, and that the former solves 97% of the instances in reasonable time when combined with a dynamic programming approach to minimize the walking time. Furthermore, once a shunting plan with a feasible staff assignment has been constructed, the MIP model can be used as a post-processing step to significantly improve the robustness of a shunting plan in one minute of computation time.

References

1 J Arturo Castillo-Salazar, Dario Landa-Silva, and Rong Qu. Workforce scheduling and routing problems: literature survey and computational study. *Annals of Operations Research*, 239(1):39–67, 2016.

2 Richard Freling, Dennis Huisman, and Albert PM Wagelmans. Models and algorithms for integration of vehicle and crew scheduling. *Journal of Scheduling*, 6(1):63–85, 2003.

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

4 Attila A Kovacs, Sophie N Parragh, Karl F Doerner, and Richard F Hartl. Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of scheduling*, 15(5):579–600, 2012.

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

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

7 Jorne Van den Bergh, Jeroen Beliën, Philippe De Bruecker, Erik Demeulemeester, and Liesje De Boeck. Personnel scheduling: A literature review. *European journal of operational research*, 226(3):367–385, 2013.

8 Roel van den Broek, Han Hoogeveen, Marjan van den Akker, and Bob Huisman. Train shunting and service scheduling: an integrated local search approach. Master's thesis, Utrecht University, 2016. URL: `https://dspace.library.uu.nl/handle/1874/338269`.

9 Roel van den Broek, Han Hoogeveen, Marjan van den Akker, and Bob Huisman. A local search algorithm for train unit shunting with service scheduling. *Transportation Science*, 2020. Manuscript submitted for publication.

A Branch-and-Price Formulations

A.1 Sequence Model

In the master problem of the *sequence model* we decide on the completion time $c_i \geq 0$ and the slack $s_i \geq 0$ of each activity $a_i \in A$. Furthermore, we have to assign the activities to staff members. We model the schedules of individual staff members as sequences of the activities in A. Let Π_{seq} be the set of all possible individual staff schedules, then we can construct a feasible staff assignment by selecting for each staff member a schedule $\pi_k \in \Pi_{seq}$ such that all activities are covered and completed before their deadline. We represent the decision of selecting staff schedule $\pi_k \in \Pi_{seq}$ by the binary decision variable y_k, where

$$y_k = \begin{cases} 1 & \text{if staff schedule } \pi_k \text{ is chosen,} \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

We denote the following properties of the staff schedule $\pi_k \in \Pi_{seq}$ as

$$a_{ik} = \begin{cases} 1 & \text{if } a_i \text{ is in staff schedule } \pi_k \\ 0 & \text{otherwise} \end{cases}$$

$$r_{ijk} = \begin{cases} 1 & \text{if } a_i \text{ directly precedes } a_j \text{ in schedule } \pi_k \\ 0 & \text{otherwise} \end{cases}$$

The problem of finding a staff assignment that maximizes the slack of the activities can then be formulated as

$$\max \sum_i s_i \qquad \text{s.t.} \tag{12}$$

$$\sum_k a_{ik} y_k = 1 \qquad \forall i \qquad (\alpha_i) \tag{13}$$

$$c_j - c_i - s_i - Q_{ij} \sum_k r_{ijk} y_k \geq D_{ij} \qquad \forall i, j \qquad (\beta_{ij}) \tag{14}$$

$$c_i + s_i \leq lct_i \qquad \forall i \tag{15}$$

$$\sum_k y_k \leq m \qquad (\gamma) \tag{16}$$

$$y_k \in \{0, 1\}, c_i \geq ect_i, s_i \geq 0. \tag{17}$$

In this model, constraint (13) ensures that all activities are performed by a staff member. The other constraints are similar to the constraints of the mixed integer linear program discussed earlier in this section. The dual variables of the constraints are denoted between the braces. The pricing problem is then to find a staff schedule that minimizes

$$\sum_i \alpha_i a_{i,k} + \sum_{i,j} \beta_{i,j} Q_{i,j} r_{i,j,k} \tag{18}$$

subject to the feasible completion time intervals $[ect_i, lct_i]$, the minimum time lag constraints between activities derived from the $\mathcal{POS}$, and the walking time of the staff member.

To solve the pricing problem, we construct the staff schedules with dynamic programming over subsequences of the priority list $L = (a_1, \dots, a_n)$. For any $a_i \in L$, we can either

1. create a schedule π^i consisting only of activity a_i, or
2. extend a schedule π' ending at activity $a_{i'}$ with $i' < i$ to schedule $\pi^{\pi' \to i}$ by appending a_i to the sequence.

The costs of these staff schedules in the pricing problem are

$$cost(\pi^i) = \alpha_i, \tag{19}$$

$$cost(\pi^{\pi' \to i}) = \alpha_i + \beta_{i',i} Q_{i',i} + cost(\pi'). \tag{20}$$

The reduced cost of a schedule π can be determined solely from the cost and last activity of the schedule π' that it extends. However, not all schedules are feasible due to deadlines and minimum time lag constraints of the activities. A schedule π is feasible if all activities can be completed on their deadline without violating the time lag constraints in the $\mathcal{POS}$ and the additional constraints resulting from π.

To verify whether schedules satisfy both the walking time and the deadline constraints, it is sufficient to keep track of the (earliest) completion times of the schedules, which we denote by $ect(\pi)$. We compute this completion time of a schedule as

$$ct(\pi^i) = ect_i, \tag{21}$$

$$ct(\pi^{\pi' \to i}) = \max\left\{ect_i, ct(\pi') + \max\{\omega_{i',i}, d_{i',i}\}\right\}. \tag{22}$$

Staff schedule π ending with activity a_i is feasible with respect to the walking time and deadline constraints if activity a_i and all its successors a_j in the $\mathcal{POS}$ can be completed before their deadlines. That is, the constraints $ct(\pi) \leq lct_i$ and, for all $j > i$, $ct(\pi) + d_{i,j} \leq lct_j$ are satisfied.

While the walking constraints only affect consecutive activities in the schedule, the $\mathcal{POS}$ can impose minimum time lag constraints on activities that are not direct successors in the schedule. As a result, storing the earliest completion time of a schedule is no longer sufficient if we want to determine whether the schedule satisfies the minimum time lag constraints as well. Therefore, we extend our characterization of a schedule π with a vector $\vec{ect}(\pi)$ of the earliest completion times of all activities a_j with $j \geq i$. With this vector we can propagate the minimum time lag constraints in the schedule and check if the schedule violates a deadline constraint. For any $j > i$, its earliest completion time with respect to the schedule is

$$\vec{ect}(\pi^i)_j = ect_j, \tag{23}$$

$$\vec{ect}(\pi^{\pi' \to i})_j = \max\left\{\vec{ect}(\pi')_j, ct(\pi^{\pi' \to i}) + d_{i,j}\right\}, \tag{24}$$

with the completion time of an extended schedule equal to

$$ct(\pi^{\pi' \to i}) = \vec{ect}(\pi^{\pi' \to i})_i = \max\left\{\vec{ect}(\pi')_i, ct(\pi') + \omega_{i',i},\right\}. \tag{25}$$

A schedule π with last activity a_i satisfies all deadline, walking and minimum time lag constraints if, for all $j \geq i$, it holds that $\vec{ect}(\pi)_j \leq ect_j$.

The pricing problem can now be solved by selecting a feasible schedule π with minimal cost. If the reduced cost of this schedule, which is $-\gamma - cost(\pi)$, is greater than zero, then the schedule is added to the restricted master problem.

We reduce the number of schedules that have to be evaluated in our dynamic programming approach by applying a domination criterion. For two schedules π_1 and π_2 ending with activity a_i, if both $cost(\pi_1) \leq cost(\pi_2)$ and, for all $j \geq i$, $\vec{ect}(\pi_1)_j \leq \vec{ect}(\pi_2)_j$, then π_1 is at least as good as π_2. This allows us to safely discard π_2 in our dynamic program.

Although the domination criterion removes many redundant schedules, the number of schedules evaluated in the dynamic programming approach can still become very large. To efficiently find a schedule with positive reduced cost or determine that no such schedule exists, we solve a relaxation of the pricing problem. In this relaxation, we only keep track of the cost and the completion time of a schedule in our dynamic program, ignoring the earliest completion time vector $\vec{ect}$, and thus the minimum time lag constraints. By simplifying the characterization of the schedules the number of potential non-dominated solutions is reduced drastically. If the reduced cost of the optimal solution to the relaxed pricing problem is non-positive, then the optimal schedule in the original pricing problem is non-positive as well. Furthermore, if the optimal solution, or any other solution constructed in the relaxation, has a positive reduced cost and is feasible with respect to the minimum time lag constraints, then we can add it to the restricted master problem. In the case that the relaxed pricing problem has solutions with positive reduced cost, but all are infeasible, then we have to solve the original pricing problem to determine if there are any schedules that can be added to the master problem.

When we cannot find any new staff schedule with positive reduced cost, then we solve the restricted master problem to optimality. In the case that the optimal solution contains fractional staff schedules, we search for an integral solution by branching on the fractional properties of the solution. We identify a pair of activities (a_i, a_j) that appears as direct successors in a fraction of the staff schedules in the optimal solution. Based on this pair we create two branches: one in which we exclude all staff schedules in which a_i directly precedes a_j, and another branch in which we exclude all staff schedules that contain a_i or a_j, but not the direct precedence relation $a_i \rightarrow a_j$. We then solve the pricing problem subject to this constraint. In general we use a best-bound search to explore the nodes, and a depth-first search if we have improved our best solution in a node.

A.2 Timestamp Model

In the *timestamp model* we do not model the completion times of the activities as decision variables. Instead, we decide on the slack times s_i and the selection of staff schedules with activity completion times z_l, where

$$z_l = \begin{cases} 1 & \text{if staff schedule } \pi_l \in \Pi_{time} \text{ is chosen,} \\ 0 & \text{otherwise.} \end{cases} \tag{26}$$

We denote for staff schedule $\pi_l \in \Pi_{time}$

$$a_l = \begin{cases} 1 & \text{if } a_i \text{ is in schedule } \pi_l \\ 0 & \text{otherwise} \end{cases}$$

$$c_{il} = \text{completion time of } a_i \text{ in schedule } \pi_l$$

$$s_{il} = \text{slack of } a_i \text{ in schedule } \pi_l.$$

The master problem of the *timestamp model* can then be formulated as

$$\max \sum_i s_i \qquad \text{s.t.} \tag{27}$$

$$\sum_k a_{il} z_l = 1 \qquad \forall i \qquad (\alpha_i) \tag{28}$$

$$\sum_k c_{jl} z_l - \sum_k c_{il} z_l - s_i \geq D_{ij} \qquad \forall i, j \qquad (\beta_{ij}) \tag{29}$$

$$\sum_k c_{il} z_l + s_i \leq lct_i \qquad \forall i \qquad (\phi_i) \tag{30}$$

$$s_i - \sum_k s_{il} z_l \leq 0 \qquad \forall i \qquad (\psi_i) \tag{31}$$

$$\sum_k z_l \leq m \qquad (\gamma) \tag{32}$$

$$z_l \in \{0, 1\}, s_i \geq 0. \tag{33}$$

The model is similar to the formulation of the master problem of the *sequence model* presented earlier. The exception is constraint (31), which ensures that the slack of activity a_i does not exceed the slack that a_i has in the staff schedules. The objective of the pricing problem of the *timestamp model* is to maximize

$$\gamma + \sum_i \left(\alpha_i + \left(\phi_i + \sum_{a_j \prec a_i \in \mathcal{POS}^+} \beta_{ji} - \sum_{a_i \prec a_j \in \mathcal{POS}^+} \beta_{ij} \right) c_i - \psi_i s_i \right) a_i. \tag{34}$$

Similar to the pricing problem of the *sequence model*, we can construct the staff schedules by dynamic programming over subsequences of the priority list $L = (a_1, \ldots, a_n)$. However, the main difference of the two pricing problems is that we have to assign completion times to the activities in the schedule as well in the *timestamp model*. Therefore, for each schedule ending with activity a_i, we have to label the completion time of a_i in the state variables of the dynamic programming algorithm to determine the optimal completion times. Due to the large number of state variables, the pricing problem of the *timestamp model* requires more computation time to construct solutions than the *sequence model* formulation. Therefore, we expect that the *sequence model* will outperform the *timestamp model*.