Single-stage shunting minimizing weighted departure times

Florian Jaehn, Johannes Rieder *, Andreas Wiehl

University of Augsburg, Sustainable Operations and Logistics, Universitaetsstr. 16, D-86159 Augsburg, Germany

ARTICLE INFO

Article history:
Received 5 March 2014
Accepted 4 November 2014
This manuscript was processed by Associate Editor Pesch.
Available online 15 November 2014

Keywords:
Single-stage shunting
Scheduling
Weighted completion time

ABSTRACT

In a traditional rail-freight hump yard, a huge number of freight cars are perpetually shunted to form outbound trains. In order to transport each car to its destination, the inbound trains are decoupled and disassembled into individual cars, which are then moved to one of the several classification tracks where they are reassembled to form new outbound trains. Motivated by the situation at Munich shunting yard, we consider a traditional single-stage shunting problem, where freight cars form new, single-destination trains with an arbitrary freight car order. There might be multiple trains to one destination so that the assignment of freight cars to outbound trains is determined by the sequence of inbound trains to be processed. Each freight car has a priority value and the objective is to minimize the weighted sum of priority values of outbound trains multiplied by the time units that have elapsed until departure. First, we elaborate a MIP formulation and then we provide a lower bound and develop precedence relations. Furthermore, we present heuristic procedures as well as a branch and bound approach. The paper concludes with computational results comparing the proposed algorithms with CPLEX.

© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

The paper on hand focuses on the development of computerized solutions for a particular problem within the area of railway optimization. Our aim is to determine a humping sequence of inbound trains. The freight cars of each inbound train are then guided to predetermined tracks, each of which corresponds to a destination. Once the outbound capacity for one destination is reached, the according freight cars form an outbound train, which immediately departs. The next freight cars to this destination must not leave until the capacity for the next outbound train for this destination is reached. The objective is to minimize the total weighted departure times, considering the weighted sum of priority values of outbound trains multiplied by the time units that have elapsed until departure. In railway terms, the generic problem is called shunting, marshaling or classification problem.

Fig. 1 shows a schematic layout of a hump yard. At the receiving tracks, there are inbound trains that contain freight cars each of which head to one destination. In order to transport each freight car to its destination, the inbound trains are decoupled and disassembled into individual freight cars. Then an inbound train is humped as a whole over the hump, whose inclination accelerates the freight cars by gravity. Via a system of tracks and switching points the freight cars are humped to given classification tracks, where they can be reassembled such that homogeneous outbound trains with freight cars in a shunting-dependent order can be generated. This process is called classification (also marshaling or shunting), and is conducted in so-called classification yards (also marshaling or shunting yards), which are intermediate stops (terminals) in the rail network. Hence, the shunting yard is the basis for a network being able to serve each origin–destination combination without providing a huge number of point-to-point connections. Finally, a train is pulled as a whole from the classification tracks to the departure tracks, where the cars are coupled, a break inspection for each car occurs, and where a locomotive is attached. Our problem is restricted to forming single-destination trains with arbitrary freight car order. Some destinations are supplied by more than one outbound train. In this case, the assignment of freight cars to the corresponding outbound trains is determined by the humping sequence. Obviously, the number of required classification tracks is at most commensurate to the number of destinations supplied by the inbound trains.

There are various papers related to the problem of shunting both freight and passenger trains. In this context, we focus on the operational processes of shunting. Strategic decisions comprise terminal location and network design/expansion (see [1,3]). On a tactical basis, important decisions are the train movements (routing) in the network (see [8,13,19]) and the distribution of locomotives or empty wagons in the network (see [17,10]). A broad survey on the operational processes is given by Boysen et al. [5] and a classification of problems with the objective of minimizing the number of classification tracks used is given by Hansmann and Zimmermann [12]. We summarize the papers closely related to the problem at hand.
The assumptions made in the paper of Yagar et al. [18] are partly related to our problem. There, a humping sequence is to be found in a static model with single-stage sorting, i.e. a situation in which freight cars must not be pulled back to the receiving tracks so that they can be humped more than once. The sequence determines whether freight cars can reach their designated outbound train or whether they are assembled to a posterior outbound train. The objective is to minimize yard throughput costs, which are influenced by the freight cars dwell time and priority values for certain freight cars. For a small number of inbound trains dynamic programming can be used for evaluation. Otherwise a ranking procedure determines a set of trains suitable for being humped next. Contrary to the problem at hand, Yagar et al. [18] consider outbound trains with a given departure time. Therefore, the proposed approach cannot be applied to our problem at hand.

An approach to reduce the dwell time for a given humping sequence has been done by Bektaş et al. [2]. They make use of the fact that for a given assignment of freight cars from inbound trains to outbound trains, two empty freight cars of the same type can be reassigned to the respective other train. Analyzing beneficial switches of empty freight cars from an inbound train (or a concatenation of inbound trains to be humped together), a maximum weighted matching problem results, which can be solved exactly using the Hungarian Method.

Another specific single-stage sorting problem has been presented by Dahlhaus et al. [9]. Their problem called “Train Marshalling Problem” can be described as follows. Freight cars from one inbound train are rearranged to form one outbound train, which serves several destinations. The order of freight cars in the outbound train must ensure that all freight cars to one destination are consecutive. Freight cars are humped to different classification tracks and on the other end of the tracks, the freight cars may be merged again to form the outbound train. The objective is to use as few classification tracks as possible since those are considered to be the bottleneck. It is proven that the problem is NP-complete and a tight upper bound is given for the case that the sequence of freight cars in the inbound train is unknown.

In a subsequent paper, Brueggeman et al. [7] provide a solution for this problem for a fixed parameter using dynamic programming. They prove that the “Train Marshalling Problem” can be solved in polynomial runtime, if the number of classification tracks is fixed.

Di Stefano and Koci [11] consider single-stage shunting problems, which originate from the parking of (passenger) trains in a depot at night. These problems can easily be reformulated to fit in the context of freight cars. For a given sequence of n inbound trains, the assignment of the trains to classification tracks needs to be determined in such a way that a given outbound sequence can be followed without extra switching. The only objective is to use as few tracks as possible.

A different setting is to be found by Bohlin et al. [4]. There, tracks in the classification yard have to be assigned to outbound trains over time. Each inbound train has an arrival time and each outbound train a departure time. Shunting freight cars from the receiving tracks to the classification tracks is called a roll-in operation and the according movement in the opposite direction is called a pull-out operation. In order to achieve the desired formation of an outbound train, pull-out and roll-in operations are used, whereby roll-in operations for inbound trains have a fixed time. The number of outbound trains is assumed to be larger than the number of available classification tracks. If a freight car is humped but none of the classification tracks is reserved for its outbound train, it is stored intermediated on a ‘mixed track’. At fixed times, all freight cars of a mixed track are pulled back and they are humped again.

Using the possibility of rehumping for a set of freight cars, Kraft [16] has developed a model, where three track-pulls are performed within 1 day. Freight cars from inbound trains dedicated to outbound trains in a free order either are considered as hard constraints or they are dropped in a preprocessing step. Using arrival, departure and processing times in a mixed integer problem formulation, the objective is to minimize the exponential lateness of all outbound trains. Furthermore, they use a truncated branch and bound algorithm solving the problem.

Recently, Jacob et al. [14] have considered a problem of multi-stage shunting, i.e. the pull back operation for all freight cars on a classification track (‘track pull’) is systematically used for ensuring a specific order of freight cars in each outbound train. For a given number of classification tracks, the objective is to minimize the number of track pulls. In order to describe a shunting schedule, the movement of each freight car in the yard has to be recorded. To do so, Jacob et al. [14] develop a novel encoding scheme for train classification schedules. For various problem settings, they either provide exact algorithms with polynomial runtime or they prove NP-hardness. A missing proof has been complemented by Briskorn and Jaehn [6].

In company with the paper on hand a different problem setting has been investigated by Jaehn et al. [15]. There, outbound trains, each of which has an arbitrary freight car order, have to be formed. The assignment of freight cars to outbound trains is given. Each outbound train has a departure time and a priority value, which is based on the priority of its freight cars. The objective is to minimize total weighted tardiness, which is measured by the multiplication of outbound priority values and delays. Having proved the problem to be NP-hard, precedence relations and optimality conditions are derived. The paper concludes with a computational study comparing results of exact and heuristic approaches.

Even though the papers above have some connections to the problem at hand, there are always major differences such that ideas for tackling those problems cannot be transferred to our problem. To the best of our knowledge, there is no algorithm proposed in the literature, which can directly be applied here.

2. Problem description

2.1. Mathematical program

In the following, we present the single-stage shunting problem minimizing weighted departure time (SSSWD) in detail and develop a static, deterministic MIP formulation. Furthermore, we prove that SSSWD is NP-hard in the strong sense and we derive a lower bound and precedence relations.

SSSWD is to find a humping sequence for T inbound trains. There are no restrictions on this sequence so that every sequence is considered to be a feasible solution. However, solutions differ in their objective function value, which is to be explained below. We call the humping of one inbound train a ‘step’ so that there are T humping steps. In total there are R freight cars in the T inbound trains. The length of inbound trains, i.e. their number of freight cars, may vary. Let us assume (w.l.o.g.) that the freight cars are numbered
from 1 to R and that freight car numbers within an inbound train are always increasing from the front of the train to the back. Let tᵢ denote the inbound train of freight car r. Each freight car r heads to a unique destination dᵢ out of D different destinations. Furthermore, each freight car has a priority value wᵢ, representing the importance of its load (in practical applications wᵢ = 0 denotes an empty freight car). Each destination d ∈ \{1,...,D\} is served by one or more outbound trains. O is the number of outbound trains and Oᵢ ⊆ \{1,...,O\} describes the set of outbound trains to destination dᵢ which leave the yard in a predefined order. Each outbound train o has a capacity of Cₒ freight cars. We assume that the number of classification tracks exceeds the number of destinations D so that a classification track is reserved for each destination. The length of each classification track is assumed to be sufficiently large. Given a humping sequence of the inbound trains, the composition and the departure times of the outbound trains can be derived as follows. All freight cars of an inbound train are humped from front to back so that each freight car is guided to the classification track of its destination. Note that the relative order of two freight cars does not change in this process. The processing time of an inbound train is calculated by a fixed setup time st plus the number of freight cars, each of which is assumed to have a processing time of one time unit. Note that thus, the processing time of a humping step is sequence dependent. If for some destination the capacity of the next outbound train is reached, this train departs immediately after the shunting process of the inbound train is completed. For safety reasons, an outbound train must not depart while shunting is in progress. However, the setup time for the next inbound train is sufficiently large so that the outbound train may leave before the shunting of the next train starts. Thus, all departure times of outbound trains correspond to the completion time of a step, i.e. a point in time in which an inbound train has completely been processed but the shunting of the next inbound train has not started yet. Obviously, two or more outbound trains may have the same departure time (even for one destination). The weight of an outbound train is defined by the sum of the weights of its freight cars. The objective is to minimize the weighted departure time of all outbound trains. This objective corresponds to the (weighted) dwell time of freight cars in the yard, which is a classical performance measure of shunting yards and is therefore of high practical relevance.

Table 1 replicates and complements the notation and in the following, we summarize the assumptions made:

1. The assignment of tracks to outbound trains can be chosen arbitrarily.
2. Each train can be shunted in each step, there are no delayed trains, no uncertainty in the input data, there are no time windows for arrival and departure.
3. There is a sufficient number of tracks: there are at least D parallel classification tracks with no length restrictions.
4. Freight cars are numbered increasingly according to their position in the inbound train.
5. There is a sequential sorting, i.e. all units of an inbound train have to be shunted before another train is processed.
6. All freight cars enter the yard from one side and leave the yard on the other side.
7. There is no restriction on the assignment of freight cars to outbound trains (but to destinations). There is no restriction on the freight car order within an outbound train, which is determined by the humping sequence.
8. The capacity of all outbound trains to one destination corresponds to the number of freight cars to this destination.
9. The time required for inspections, uncoupling, assembly is expressed by the fixed setup time st.

We developed different MIP-formulations for the problem described above. Some were based on a time discretized decision variables and others on sequence representing decision variables. After comparing those formulations concerning the number of variables and the number of constraints, we consider the following formulation to be best:

Minimize  \( \sum_{o=1}^{O} \sum_{r=1}^{R} \sum_{t=1}^{T} y_{r,s,o,a} \cdot p_a \cdot w_r \)  
subject to \( \sum_{t=1}^{T} x_{t,s} = 1 \) \( \forall s = 1...S \)  
\( \sum_{t=1}^{T} \sum_{b=1}^{b} (R_{C,t} \cdot x_{t,b}) - \sum_{j,t}^{T} \sum_{j,t}^{T} C_o \geq (v_{s,o} - 1) \cdot M \) \( \forall s = 1...S; d = 1...D; o \in O_d \)  
\( z_{r,s,o} \leq v_{s,o} \) \( \forall r = 1...R; s = 1...S; o \in O_d \)  
\( \sum_{s=1}^{S} \sum_{o \in O_d} z_{r,s,o} = 1 \) \( r = 1...R \)  

Table 1

Notation: indices, parameters and variables.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T: S; A; B</td>
<td>Number of inbound trains (indices t, s, a, b)</td>
</tr>
<tr>
<td>D</td>
<td>Number of destinations (index d)</td>
</tr>
<tr>
<td>R, L</td>
<td>Number of freight cars (indices r, l)</td>
</tr>
<tr>
<td>tᵢ</td>
<td>Inbound train of freight car r; tᵢ ∈ {1,...,T} ( \forall r \in {1,...,R} )</td>
</tr>
<tr>
<td>dᵢ</td>
<td>Destination of freight car r; dᵢ ∈ {1,...,D} ( \forall r \in {1,...,R} )</td>
</tr>
<tr>
<td>wᵢ</td>
<td>Priority weight of freight car r; wᵢ ∈ {0, 1} ( \forall r \in {1,...,R} )</td>
</tr>
<tr>
<td>RCᵢ,t</td>
<td>Number of freight cars from inbound train t to destination dᵢ; ( \forall t \in {1,...,T} )</td>
</tr>
<tr>
<td>O,E</td>
<td>Number of outbound trains (indices o, e)</td>
</tr>
<tr>
<td>Cₒ</td>
<td>Capacity of outbound train o; ( \forall o \in {1,...,O} )</td>
</tr>
<tr>
<td>C_max</td>
<td>Maximum capacity of any outbound train, ( C_{\text{max}} = \max_{o=1...O} C_o )</td>
</tr>
<tr>
<td>O_d</td>
<td>Set of outbound trains to destination d, O_d ⊆ O</td>
</tr>
<tr>
<td>dᵢ</td>
<td>Destination of outbound train o; ( \forall dᵢ \in {1,...,D} )</td>
</tr>
<tr>
<td>st</td>
<td>Setup time of an inbound train in time units</td>
</tr>
<tr>
<td>pᵢ</td>
<td>Processing time of inbound train t in time units, ( p_i = st + \sum_{t=1}^{T} RC_{i,t} )</td>
</tr>
<tr>
<td>M</td>
<td>Big integer (e.g. ( M = C_{\text{max}} \cdot \max_j (</td>
</tr>
<tr>
<td>xᵢ,s</td>
<td>Binary variable: 1, if train t is humped at step s; 0, otherwise</td>
</tr>
<tr>
<td>qᵢ,a</td>
<td>Binary variable: 1, if inbound train t is humped before inbound train a; 0, otherwise</td>
</tr>
<tr>
<td>vᵢ,s,o</td>
<td>Binary variable: 1, if outbound train o departs in an earlier step s′ &lt; s; 0, otherwise</td>
</tr>
<tr>
<td>zᵢ,s,o</td>
<td>Binary variable: 1, if freight car r is assigned to outbound train o ∈ O_d, which departs immediately after step s; 0, otherwise</td>
</tr>
<tr>
<td>yᵢ,s,o,a</td>
<td>Binary variable: 1, if freight car r is assigned to outbound train o ∈ O_d, which departs no later than in step s; 0, otherwise</td>
</tr>
</tbody>
</table>
Objective function (1) minimizes the weighted departure times of all outbound trains. Therefore, we consider all freight cars of all outbound trains, as the weighted departure time of an outbound train can be derived as the sum of all its freight car weights. For each inbound train not having departed later than the considered freight car, the processing time is summed up. This value is multiplied by the priority value of the considered freight car. Doing so for all freight cars, inbound trains, and steps we obtain the weighted departure time for each outbound train. Finally the values have to be summed up for all outbound trains and destinations.

Equalities (2) ensure that in each step only one inbound train is humped. Simultaneously, each train can only be humped once, which is ensured in the course of the model by (5). To ensure that an outbound train only departs when its capacity limit is reached, we define binary variable \( v_{o,r} \). Note that several outbound trains to one destination depart in increasing order in terms of their values \( o \). The objective will always try to set \( v_{o,r} \) to one so that we only have to force its value to be zero, whenever the outbound train \( o \) has departed before step \( s \). Obviously, this is the case if and only if the capacity of the considered outbound train has been reached, which is guaranteed by (3). In the following, we introduce the binary variable \( z_{r,s,o} \). There are four necessary restrictions to be formulated. Obviously, \( z_{r,s,o} \) is related to \( v_{o,r} \) (4). Here, the best objective function value would be obtained if no \( z_{r,s,o} \) is set to one. We prevent this case by formulating a “flow maintenance restriction” (5), which ensures that each freight car is assigned to one outbound train, which leaves in exactly one step. Restriction (6) incorporates the limited capacity of an outbound train. We still have to restrict \( z_{r,s,o} \) regarding the order of the freight cars, i.e. a freight car shunted later than another one must not leave on an earlier outbound train. As within one inbound train, freight cars with lower index are shunted first, their respective order in the outbound train (if they head to the same destination) does not change. Thus, for two freight cars \( r \) and \( l \) with \( r < l \) from the same inbound train \( t_{r} = t_{l} \) must not be assigned to an outbound train leaving earlier than the outbound train \( r \) is assigned to. This is ensured by (7). There is another restriction needed on the freight car order implied by the humping sequence. If an inbound train \( t \) is humped before inbound train \( a \), then for each destination, freight cars from \( t \) are never assigned to an outbound train with a higher index than those of the inbound train \( a \). This is ensured by (9), which makes use of binary variable \( d_{t,a} \). If an inbound train \( t \) is humped before another inbound train \( a \), \( q_{o,a} \) obtains the value 1. If inbound train \( a \) is humped at the current step and \( t \) is also humped up to the current step, \( t \) has been humped in a previous step. Thus, \( q_{o,a} \) has to obtain the value one (8). It is redundant to force \( q_{o,a} \) to be zero whenever \( a \) is humped before \( t \), as in this case, (9) is looser. Furthermore, the restriction is only relevant if both freight cars depart to the same destination. Finally, we introduce binary variables \( y_{r,s,o,a} \), which determine the objective function value. It is strongly related to \( z_{r,s,o} \), obtaining the same value if inbound train \( a \) is humped no later than inbound train \( t \). Note that we have to include the case \( a = t \). Constraint (10) forces \( y_{r,s,o,a} \) to be one if \( z_{r,s,o} = 1 \) and if \( a \) is humped no later than \( t \). Otherwise, the objective function ensures that \( y_{r,s,o,a} \) takes value zero. The binary variable \( y_{r,s,o,a} \) not only contains information about the assignment of freight cars to outbound trains, which is determined by the humping sequence of inbound trains, but it also indirectly contains information about the time passed by. Since we know the processing time of each inbound train, the time value of a special step is the result of the sum of all trains having been humped no later than this step. Finally, constraints (11) ensure that the binary variable may only take values one or zero. Note that these are the only constraints, which are not linear.

2.2. Complexity and bounds

**Theorem 2.1 (Complexity).** The SSSWD is strongly \( NP \)-hard.

**Proof.** By reduction from strongly \( NP \)-hard 3-Partition, which can be described as follows. Given \( 3q+1 \) positive integer numbers \( h_{1}, \ldots, h_{3q} \) and \( H \) such that \( H/4 < h_{i} < H/2 \), \( i = 1 \ldots 3q \), and \( \sum_{i=1}^{3q} h_{i} = qH \). Is there a partition of the set \( \{1 \ldots 3q\} \) into \( q \) disjoint sets \( X_{1} \ldots X_{q} \) such that \( \sum_{i \in X_{l}} h_{i} = H \) for \( l = 1 \ldots q \)?

From an instance of 3-Partition, we construct the following instance of SSSWD. There are \( 3q \) inbound trains and \( R = qH \) freight cars. All freight cars go to a single destination and all freight car weights are equal. Furthermore, \( RC_{i} = h_{i} \), \( i = 1 \ldots 3q \) holds, i.e. the number of freight cars of inbound train \( i \) corresponds to the \( i \)th given number of 3-Partition. There are \( O = q \) outbound trains, each of which has capacity \( C_{o} = H \), \( o \in \{1 \ldots q\} \). The setup time for inbound trains is assumed to be zero.

We will now show that if and only if we have a YES-instance of 3-Partition, the corresponding instance of SSSWD has an optimum objective function value of \( (q^{2}+q)/2H \). Note that \( (q^{2}+q)/2H \) is a lower bound for the objective function value of SSSWD, as the \( o \)th outbound train \( (o = 1 \ldots q) \) must not depart before time \( oH \) because otherwise, the train’s capacity would not have been reached.

Assume that a YES-instance of 3-Partition is given. Consider a solution of SSSWD in which all sets of trains corresponding to a disjoint set of the 3-Partition solution are shunted consecutively. As there is no setup time, the \( H \)th freight car, which is shunted, is the last one of an inbound train. Thus, the first outbound train departs at point in time \( H \). Similarly, the next outbound train departs at \( 2H \), and so on. Thus, the instance of SSSWD has an optimum objective function score of \( (q^{2}+q)/2H \).

Now assume for a contradiction that we are given a NO-instance of 3-Partition, but the according instance of SSSWD has an optimum objective function score of \( (q^{2}+q)/2H \). As described above, this objective function score can only be reached if the \( o \)th outbound train leaves at point in time \( oH \). No outbound train can depart earlier, because in this case, not enough freight cars would be available. An outbound train \( o \) certainly departs later than \( oH \), if the humping of an inbound train is not completed exactly at time \( oH \). So we may assume that an inbound train’s processing stops at \( oH \) for all \( o = 1 \ldots q \). However, this implies that there is a partition of the set \( \{1 \ldots 3q\} \) into \( q \) disjoint sets \( X_{1} \ldots X_{q} \) such that \( \sum_{i \in X_{l}} h_{i} = H \) for
In the proof we use a reduction to instances of SSSWD in which all priority weights are equal, in which there is only one destination, in which all outbound train capacities are equal, and with zero setup time. Thus, it follows that even with these restrictions SSSWD remains strongly NP-hard.

We now develop a lower bound for the objective function score of SSSWD. Note that this lower bound can also be applied to each subproblem in which the assignment of inbound trains is only fixed for the first \( b, b \in \{0 \ldots T \} \) steps. To ease reading, we limit our explanations of the lower bound to the whole problem with \( b=0 \), though the specifications for subproblems should be obvious. For each destination \( d \) and each outbound train \( o \) heading to this destination, we identify the earliest step in which \( o \) may depart. Therefore, we simply determine the set of inbound trains \( I(d) \) that have freight cars dedicated to \( d \). If we sort this set non-increasingly by \( RC_{i,d} \), we easily obtain the minimum number of inbound trains required to let \( o \) depart.

Similarly, we may obtain an earliest completion time of a time step, i.e. the earliest departure time of outbound trains departing after this step. Therefore, we sort \( l(d) \) non-decreasingly concerning \( p_i \), i.e. the shortest trains are first. As we know the earliest time step for an outbound train to depart, we can derive the earliest departure time \( c(o) \) for outbound train \( o \) by summing up the according processing times.

Finally, we take the priority values into account. Again, we sort set \( l(d) \), this time non-increasingly concerning the priority values \( w^d(i) \) of its freight cars heading to destination \( d \), i.e. \( w^d(i) = \sum_{t \in \{1 \ldots D \}} w_t \) with \( i \in l(d) \) and \( d = 1 \ldots D \). Again, we know the earliest time step \( s^*(o) \) for an outbound train \( o \) to depart and thus, by summing up the priority values of the first \( s^*(o) \) trains of the sorted list, we get an upper bound of the priority values of the freight cars that have departed by that step.

The relaxations we use for obtaining the lower bound can be seen as “cherry picking”, a term which is often used in connection with shunting operations. Only the lowest number of inbound trains with the lowest number of freight cars and the highest priority values are chosen in order to let an outbound train depart.

**Property 2.2 (Lower bound).** Given a destination \( d \in \{1 \ldots D \} \) and let \( o \) be the \( b \)th, \( b \in \{1 \ldots |O_d| \} \) train to destination \( d \). Let \( l(d) = \{ t \in \{1 \ldots T \} | RC_{t,d} > 0 \} \) and sort this set non-increasingly by \( RC_{t,d} \). Let \( l(d) \) denote the \( i \)th element of this list. Let \( s^*(o) \) be the minimum integer \( s \) such that \( \sum_{t \in l(d)} RC_{t,d} \geq b \cdot C_o \). \( s^*(o) \) is obviously the earliest step in which \( o \) can depart.

Sort \( l(d) \) non-decreasingly concerning \( p_i \), and let \( l(d) \) denote the \( i \)th element of this list. Then \( c(o) = \sum_{t \in l(d)} p_t \) is a lower bound for the departure time of train \( o \).

Let \( w^d(i) = \sum_{t \in \{1 \ldots D \}} w_t \) be the priority values of the freight cars of inbound train \( i \) going to destination \( d \). Sort \( l(d) \) non-increasingly by \( w^d(i) \) and let \( l(d) \) denote the \( i \)th element of this list. For the \( b \)th outbound train \( o_b \) to destination \( d \), \( w(o_b) = \sum_{t \in l(d)} w^d(i) \) is an upper bound of the priority weights of the freight cars of the first \( b \) outbound trains. Additionally, let \( W^o(0) = W^o(1) = w(o_1) = w(o_b) - w(o_{b-1}) \) for all \( b = 2 \ldots |O_d| \):

\[
LB = \sum_{o=1}^{|O_d|} c(o) \cdot w(o) \text{ is a lower bound of the objective function score of SSSWD}
\]

**Property 2.3 (Precedence relation).** For an inbound train \( t \) let \( D(t) = \{ d \in \{1 \ldots D \} | RC_{t,d} > 0 \} \) be the set of destinations served by this inbound train. Given an inbound train \( t \) with \( \max(|O_d| | RC_{t,d} > 0) = 1 \), i.e. an inbound train that only serves destinations, which have at most one outbound train. If there is a second inbound train \( a \) with \( D(t) \subseteq D(a) \), then there is an optimal solution in which \( a \) precedes \( t \).

**Proof.** Assume for a contradiction that \( t \) precedes \( a \) in any optimal solution. Obviously, all outbound trains of \( D(t) \) must not depart after the processing of \( a \). Thus, if there are any inbound trains scheduled in between \( t \) and \( a \), the objective function score does not increase if we postpone \( t \) such that it is processed immediately before \( a \). We may therefore assume that no other inbound trains are scheduled in between \( t \) and \( a \). If we then switch the positions of \( t \) and \( a \), still no outbound train departs later, as both, \( t \) and \( a \) serve every destination of \( D(t) \). Thus, we generated an optimum solution in which \( a \) precedes \( t \), which contradicts our assumption.

**Property 2.4.** Given an optimal solution \( x^* \) of an instance of SSSWD, an inbound train \( t^* \) serving just destinations with at most one outbound train \( \{O_d | RC_{t^*,d} > 0 \} = 1 \) and let \( s^* \in \{2 \ldots S \} \) be a step such that

- \( x^*_t = 1 \) for some \( s < s^* \), and
- \( \sum_{t=1}^{|S|} RC_{t,d} \cdot \sum_{s^* = s}^{S} I(s^*) \geq 1 \) for all \( d \in D(t^*) \).

\( x^*_t \) is scheduled before step \( s^* \) and all outbound trains receiving freight cars from \( t^* \) do receive further freight cars from inbound trains scheduled after \( s^* \). Consider solution \( x' \), which is derived from \( x^* \) by postponing inbound train \( t^* \) from its current step \( s^* \) to step \( s^* \) and advancing all affected inbound trains by one step, i.e.

\[
\begin{align*}
x'_{t,s} & = x_{t,s} & \text{if } s < s^* \leq s^* \\
x'_{t,s^*} & = x_{t,s^*} & \text{if } s \leq s^* \leq s^* \text{ for all } t \in \{1 \ldots T\}.
\end{align*}
\]

Then, \( x' \) is an optimal solution.

**Proof.** We will show that no outbound train departs later in \( x' \) than in \( x^* \). \( x^* \) only serves destinations with one outbound train and by our assumptions, the departure steps are always greater than \( s^* \) in solution \( x^* \). As \( x^* \) and \( x' \) do not differ starting from step \( s^* \), the previous claim also holds true for \( x' \). Thus, postponing \( t^* \) has no negative effect on the departure times. Obviously, the advance- ment of trains in the schedule cannot delay the departure of outbound trains. Thus, \( x' \) must be optimal as well.

**3. Solution algorithms**

In the following, we will present two heuristics and an exact method for solving SSSWD. For a given sequence \( \pi \), let \( f(\pi) \) denote the objective function value. The first algorithm sorts the trains in a non-increasing order concerning its weight per freight car. Therefore each train is weighted according to its priority value and its number of freight cars. The simple idea is to bump inbound trains first if the average weight of its freight cars is high such that these freight cars may depart as possible.

**Algorithm 1.** Sorting algorithm.

1. Determining a sorted sequence: Sort the set of trains non-increasingly by

\[
\frac{\sum_{t=1}^{|S|} RC_{t,d}}{\sum_{t=1}^{|S|} t \cdot w_t} \forall t = 1 \ldots T
\]

Based on an initial solution obtained from Algorithm 1, we want to improve our objective function value. We determine multiple swaps by randomly picking \( N \) different values out of the set \( \{1 \ldots T\} \), which correspond to the trains to be exchanged. All possible combinations of exchanging these \( N \) trains are then tested, i.e. there are three combinations for double swaps and 15 combinations for triple swaps.
As the number of all possible swaps is very large, we omit evaluating all of them, but restrict ourselves to a limited number of R randomly generated swaps. To increase solution quality while simultaneously reducing runtime we implement a tabu list for combinations of random numbers already having been drawn. Thus, the number of computational steps for N drawn random numbers and R repetitions is \( \sum_{r=1}^{R} \binom{N-1}{2} (2k-1) \) \( \forall r \leq \frac{k}{6} \). Afterwards, we explore the complete neighborhood, which is obtained by swapping exactly two inbound trains.

It seems noteworthy that RH performs best in a configuration with multiple swaps being carried out before having achieved good solution quality. Thus, we explore the neighborhood for two inbound trains last in each iteration. Contrary to our initial assumption that multiple swaps should be performed to overcome a local optimum, it turned out to be quite promising to approximate to the optimum by performing multiple swaps and afterwards successively exploring the neighborhood of exchanging two trains. This is due to the fact that the solution space initially is investigated to a higher extend. The probability of reaching the solution area leading to the overall optimum decreases if the multiple swaps are performed when already being trapped in a local optimum. Afterwards, Step 2 determines a local optimum, which of course could be the global optimum.

Of course, there are different methods of calibrating the algorithm. In general, the neighborhood of simple swaps will certainly deliver promising results in a very short time. Thus, the number of loops in Step 1 could even be set to zero. However, if this value is set to a positive number, solution quality is likely to increase at the cost of higher computation time. It should be noted that it might be reasonable to explore the respective neighborhood systematically instead of randomly, if a very high number of loops is desired in Step 1.

Motivated by the results of Section 2, we have developed a branch and bound algorithm. The initial branching is performed by assigning T different trains to the first available step. Thus, each node has up to T branches, at which T denotes the remaining number of available steps. The algorithm uses depth first search, always branching the higher prioritized inbound trains (relating to the sorting of Algorithm 1) first. We start branching in step 1 and continue successively to step 7. The solution is displayed using a stack I'. The operations “push” and “pop” add and remove an element from the stack, respectively. For simpler presentation purposes precedence relations (Property 2.3) are not presented in the algorithm, although the branching in Step 2 is only performed if it does not violate a precedence relation.

Algorithm 2. Randomized Hillclimber.

0. Initialization:

Determine a set of trains \( \pi \) according to the FIFO algorithm. Determine an upper bound UB of the objective function score using \( \pi \). Let \( R=200 \) be the number of repetitions. Let \( \zeta \) denote a tabu list. Let \( N=\pm 2k \).

1. Multiple random swaps:

   for \( r \) in \( R \) do
   if \( (r=(\frac{i}{j})) \) then
     | set \( N=N-2 \), clear(\( \zeta \)) and go to Step 1.
     create a set RAND of \( N \) pairwise different random numbers from the set \( \{1...T\} \).
     push (RAND) to \( \zeta \).
     for all perfect matchings of RAND do
     swap the positions of each pair in the matching in \( \pi \).
     if \( f(\pi) \leq UB \) then
     | UB = \( f(\pi) \)
     else
     | swap the positions of each pair in the matching in \( \pi \).
   end
   end
   N = N - 2
   if (N=0) then
     | go to Step 2.
   end
   else
     | clear(\( \zeta \)) and go to Step 1.
   end

2. Neighbor solutions:

   for all pairs \( (i,j), 1 \leq i < j \leq T \) do
   swap positions i and j in \( \pi \).
   if \( f(\pi) \leq UB \) then
   | UB = \( f(\pi) \)
   else
   | swap positions i and j in \( \pi \).
   end
   end
   if \( r=0 \) then
     | Stop

0. Initialization:
Determine a (global) upper bound \( UB \) of the objective function score using the solution \( x^* \) of Algorithm 2. Sort the inbound trains using Algorithm 1, store them in a list \( \mathcal{V} \) and create an empty stack \( \Gamma \) of inbound trains. Let \( f(\Gamma) \) denote the objective function score induced by the inbound trains contained in \( \Gamma \).

1. Fathoming Nodes

\[
\text{if } (|\Gamma| = T) \text{ then } \\
\text{if } (f(\Gamma) < UB) \text{ then } \\
|x^*| \equiv \Gamma, \ UB = f(\Gamma), \ \text{pop}(\Gamma) \text{ and return. \text{ end}} \\
\text{end} \\
\text{calculate } LB \text{ using Property 2.2;} \\
\text{if } (LB + f(\Gamma) \geq UB) \text{ then } \\
| \text{pop}(\Gamma) \text{ and return.} \\
\text{end}
\]

2. Branching Nodes

\[
\text{for } (i \in \mathcal{V}, \Gamma) \text{ do} \\
| \text{push}(\Gamma, i) \text{ and go to step 1.} \\
\text{end}
\]

It is also possible to change the branch and bound algorithm such that the sequence of inbound trains is determined starting with the last step, allowing us to make use of Property 2.4. We will call this variant Algorithm 4. However, as we will see in Section 4, this is not beneficial if \( T > D \). This is due to the fact that the lower bound (Property 2.2) is significantly weaker as no time values are charged for the first steps. In node \( T \), we do not know which train is humped in step 1. Thus, we have the choice between a weak current result with a strong lower bound or a strong current result with a weak lower bound. If \( T < D \), Property 2.4 can be used better and Algorithm 4 outperforms Algorithm 3.

4. Computational results

4.1. Instances generation

All methods presented in the previous section have been implemented in Java and run on an AMD FXtm-8120 Eight-Core Processor, 3.10 GHz PC, with 16 GB of memory. Further, we stored the instances data and results in a “5.6.13 MySQL Community Server" database. The instances are generated within a PHP based webservice connected to the ‘Random.org’ API for automated clients, which creates truly random numbers via atmospheric noise. To develop instances and for the computational study the following attributes are taken into account:

- Number of inbound trains.
- Processing time and setup-time.
- Number of outgoing destinations (total and per train).
- Number of freight cars per train.
- Capacity of outbound trains.
- Distribution of three priority levels.

Based on the situation at the yard “München Nord Rangierbahnhof”, we used the following information: on an usual day, there is a throughput of 1600 up to 2000 freight cars. Thus, up to 50 inbound trains arrive daily. In the course of a day there are rather few inbound trains arriving before evening. Then, there is a peak time until late at night. Hence, it seems reasonable to vary the number of inbound trains up to a limit of 40. To hump a complete inbound train requires about 20 min. In this process 5 min are needed for inspections and decoupling and the rest of it for shunting the freight cars. Consequently, we decided to fix the set-up-time in proportion of 15:1, whereby one time unit denotes 20 s. “München Nord Rangierbahnhof” is connected to 36 destinations. Among them, 14 routes serve rather big yards in Germany. The use of our model in practice is most appropriate for a hub and spoke network of hump yards, as outbound trains supplying those destinations usually are single destination trains. Hence, we chose the value 40 freight cars per outbound train. This is due to the fact that trains in Germany may not exceed a length of 740 m. Based on the average of a 17 m long freight car, a capacity of 40 seems reasonable. Finally, we have a look on the distribution of the priority levels and their values. According to data from the Canadian National Railway Company [2], about half of the freight cars are empty wagons and therefore obtain the value zero. Another half carries freight, whereof in total 10% have to depart as fast as possible and obtain the value 10. The rest is crucial as well and obtains the value 1.

Generating instances we chose the following methods and assumptions: we developed two sets of instances, which we call “SMALL” and “LARGE”. In each instance, the priority levels are uniformly distributed according to the above-mentioned values. Varying the number of inbound trains, SMALL comprises instances with 6, 7, ..., and 14 trains, and the instances of test set LARGE have 15, 18, 21,...,39 trains. The total number of destinations of an instance varies from 4, 6, 8 and 10 (SMALL) or 5, 8, 11 and 14 (LARGE). The number of destinations served by one inbound train is uniformly drawn out of the interval from 2 and \( \lfloor 2 \cdot \sqrt{D} \rfloor \). For each setting in SMALL we generate 20 instances and in LARGE we generate 10 instances. Building instances according to a multifactorial design the number of instances per set is determined by the number of trains \( T \), the number of destinations \( D \) and the number of instances. Thus, SMALL consists of 9 · 4 · 20 instances and LARGE consists of 9 · 4 · 10 instances, which results in a total test bed of 1080 instances (see Table 2).

4.2. Results

First, solution performance for instances SMALL, i.e. 720 exactly solved instances is evaluated. The results for Algorithms 1–3 (Weighted Priority (WP), Randomized Hillclimber (RH), and Branch and Bound (B&B)) are shown in Table 3. Some instances with 14 inbound trains required up to 4 h for being solved by the branch and bound algorithm. On average, we need 10.5 min to solve an instance with 13 inbound trains. Thus, we may consider 13 inbound trains to be maximum in our setting for solving SSSWD to optimality in acceptable runtime. In total, we found

<table>
<thead>
<tr>
<th>Sym.</th>
<th>Description</th>
<th>SMALL</th>
<th>LARGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T )</td>
<td>Number of trains</td>
<td>6, 7, ..., 14</td>
<td>15, 18, 21, ..., 39</td>
</tr>
<tr>
<td>( D )</td>
<td>Number of destinations</td>
<td>4, 6, 8, 10</td>
<td>5, 8, 11, 14</td>
</tr>
<tr>
<td>( d )</td>
<td>Destinations within an inbound train</td>
<td>U(2, ( \lfloor 2 \cdot \sqrt{D} \rfloor ))</td>
<td>U(2, ( \lfloor 2 \cdot \sqrt{D} \rfloor ))</td>
</tr>
<tr>
<td>( R )</td>
<td>Number of freight cars per train</td>
<td>NormDis(30,5)</td>
<td>NormDis(30,5)</td>
</tr>
<tr>
<td>( w_c )</td>
<td>Priority weight of a freight car</td>
<td>(0,1,10)</td>
<td>(0,1,10)</td>
</tr>
<tr>
<td>( C_o )</td>
<td>Capacity of outbound train</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>( s_t )</td>
<td>Setup time</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>( p_t )</td>
<td>Processing time</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
147 precedence relations in 107 instances. Unfortunately, from the instances with 12–14 trains only 11 had precedence relations with an average time saving of 153 s. On average, we save 17.5 s of time in 107 instances by using the precedence relations of Property 2.3. A main reason for the performance of our branch and bound algorithm is the low deviation of the sum of priority values among inbound trains. Nodes are fathomed rather lately and improvements are expected if the priority values of inbound trains differ to a higher extend.

Running the Randomized Hillclimber algorithm, we chose \( R = 200 \) iterations and \( N = 10 \) swap pairs. Having compared various configurations, this one seems to be the most appropriate one considering solution quality and runtime. Obviously FIFO and WP are clearly outperformed. The Randomized Hillclimber approach even solves about 58% of the instances exactly. However, with a rising number of inbound trains, the number of instances solved to optimality (SO) decreases. Compared to branch and bound, there is a relative average gap of 0.44% of the exact objective function value. Surprisingly, the gap just rises slightly with a rising number of trains. Thus, for (6–8) there is a gap of 0.38%, for (9–11) 0.44% and for (12–14) 0.46%.

Let us now consider the effects of changing one factor ceteris paribus. An increasing number of inbound trains leads to an exponentially increasing runtime of the branch and bound algorithm. An increasing number of freight cars causes more loops. Hence, it also increases runtime. For both, low values increase the possibility of finding precedence relations, as the possibility of finding destinations not exceeding the capacity limit increases. Finally, we want to consider the effect of raising the total number of destinations. Clearly, the lower bound loses strength, since less inbound trains have a freight car for a certain destination. In contrast, the number of precedence constraints rises, as high values increase the possibility of finding destinations, which do not exceed the capacity limit. Depending on the value, the possibility of finding a precedence relation will decrease again, as no matching subset might be found. This can be seen in Table 4, where set SMALL is sorted by destinations. The number of precedence relations is expressed by the abbreviation Prec. Taking a look at Table 4, one can clearly see that the number of precedence relations being found rises with an increasing number of destinations. Nevertheless, runtime increases as the lower bound loosens strength.

Furthermore, we have modeled SSSWD in ILOG and applied CPLEX 12.3 for solving the instances. Unfortunately, we had to stop the test run after some easy instances, as up to 3 h are needed to solve an instance. Runtime is much higher than it is using branch and bound and CPLEX needs a substantial amount of memory. In the interest of comparability we created 50 instances, where each of them consists of three destinations, four inbound trains, 30 freight cars and one or two destinations per inbound train. While the runtime of Branch&Bound is negligibly low, CPLEX needs 51 s on average to solve an instance.

In addition we want to show the effect of Algorithm 4 and Property 2.4. Therefore, we took a lower bound where all freight cars, which have not departed yet are assumed to depart in step 1 and thus, dropped Property 2.2 for Algorithm 4. We generated 100 instances with 10 inbound trains and 15 destinations. The other values are set according to the assumptions made above. The values are compared with Algorithm 3. On average, we save 0.63 s of time with 62 instances being solved faster. As there are more destinations than inbound trains, Property 2.4 becomes more applicable. If the assignment of inbound trains to outbound trains was given, Property 2.4 would be quite powerful.

Finally, we want to compare different configurations of RH in the environment of LARGE. Therefore, we denote the configuration as RH (NR) in Table 5. Still, runtime of the sorting algorithm is negligibly low. Due to a rising number of computational steps, the runtime of RH increases with increasing \( N \) or \( R \). Especially in case of rising \( N \) the matching pairs of RH strongly increase. Obviously, solution quality also increases.

<table>
<thead>
<tr>
<th>Number of destinations</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>B&amp;B Obj.</td>
<td>21,076,652</td>
<td>21,761,807</td>
<td>23,412,292</td>
<td>23,796,654</td>
</tr>
<tr>
<td>CPU</td>
<td>16,026.2 s</td>
<td>53,006.7 s</td>
<td>93,464.2 s</td>
<td>155,566.4 s</td>
</tr>
<tr>
<td>Prec</td>
<td>6</td>
<td>25</td>
<td>50</td>
<td>66</td>
</tr>
<tr>
<td>Sum</td>
<td>112,123,372</td>
<td>107,135,843</td>
<td>90,047,405</td>
<td>125,065.8</td>
</tr>
<tr>
<td>Mean</td>
<td>157,920.2</td>
<td>150,895.6</td>
<td>125,617.1</td>
<td>154,895.6</td>
</tr>
</tbody>
</table>

### 5. Conclusion and outlook

The paper on hand considers a specific sequencing problem in the context of humping yards. Our aim is to find a sequence of inbound trains minimizing the weighted departure time of outbound trains. Freight cars from inbound trains are therefore moved to a classification track, where they are combined to form new, single-destination outbound trains. We develop a MIP formulation and we derive lower bound arguments and precedence relations, which are applied in an exact branch and bound algorithm. In a comprehensive computational study a heuristic approach, sorting algorithms, exact branch and bound algorithms, and CPLEX are compared with each other. We give recommendations, how branch and bound should be controlled considering the problem setting.

The application of SSSWD in practice seems to be the next step for future research. While the adaptation of some further practical aspects might ease solving the problem, others may deliver further research challenges. An example of the former is the integration of arrival times, which restrict the assignment of inbound trains to...
certain steps. Considering such constraints will speed up all algorithms presented in this paper. However, SSSWD can be seen as a subproblem in the operational processes in a rail network with many interactions to other optimization problems such as train routing or timetabling. Focusing on these interactions certainly delivers further research directions of high practical interest.

References


Table 5
Comparison of the methods on 3 · 120 heuristically solved instances (LARGE).

<table>
<thead>
<tr>
<th>Number of inbound trains</th>
<th>Sum Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>(15–21)</td>
<td>(24–30)</td>
</tr>
<tr>
<td>(33–39)</td>
<td></td>
</tr>
<tr>
<td>FIFO</td>
<td>Obj. 56,553,753</td>
</tr>
<tr>
<td>WP</td>
<td>Obj. 52,764,361</td>
</tr>
<tr>
<td>RH(6,150)</td>
<td>Obj. 45,252,924</td>
</tr>
<tr>
<td>RH(8,150)</td>
<td>CPU 30.7 s</td>
</tr>
<tr>
<td>RH(6,300)</td>
<td>Obj. 44,970,686</td>
</tr>
<tr>
<td>CPU 84.4 s</td>
<td>226.8 s</td>
</tr>
<tr>
<td>RH(10,150)</td>
<td>Obj. 44,840,028</td>
</tr>
<tr>
<td>CPU 760.3 s</td>
<td>1265.0 s</td>
</tr>
<tr>
<td>RH(6,300)</td>
<td>Obj. 45,148,841</td>
</tr>
<tr>
<td>CPU 40.6 s</td>
<td>139.3 s</td>
</tr>
<tr>
<td>RH(8,300)</td>
<td>Obj. 44,865,689</td>
</tr>
<tr>
<td>CPU 1476.6 s</td>
<td>340.4 s</td>
</tr>
<tr>
<td>RH(10,300)</td>
<td>Obj. 44,698,098</td>
</tr>
<tr>
<td>CPU 40.6 s</td>
<td>139.3 s</td>
</tr>
</tbody>
</table>