Jena Research Papers in Business and Economics

Scheduling freight trains in rail-rail transshipment yards

Nils Boysen, Erwin Pesch

11/2008

Jenaer Schriften zur Wirtschaftswissenschaft

Working and Discussion Paper Series
School of Economics and Business Administration
Friedrich-Schiller-University Jena

ISSN 1864-3108

Publisher:

Wirtschaftswissenschaftliche Fakultät
Friedrich-Schiller-Universität Jena
Carl-Zeiß-Str. 3, D-07743 Jena

www.jbe.uni-jena.de

Editor:

Prof. Dr. Hans-Walter Lorenz
h.w.lorenz@wiwi.uni-jena.de

Prof. Dr. Armin Scholl
armin.scholl@wiwi.uni-jena.de

www.jbe.uni-jena.de

Scheduling freight trains in rail-rail transshipment yards

Nils Boysen
Lehrstuhl für Allgemeine Betriebswirtschaftslehre / Operations Management
Friedrich-Schiller-Universität Jena, Carl-Zeiß-Straße 3, D-07743 Jena, Germany
nils.boysen@uni-jena.de

Erwin Pesch
Institut für Wirtschaftsinformatik, Universität Siegen, Hölderlinstraße 3, D-57068 Siegen, Germany
erwin.pesch@uni-siegen.de

May 22, 2008

Abstract

Transshipment yards, where gantry cranes allow for an efficient transshipment of containers between different freight trains, are important entities in modern railway systems and facilitate the general shift from point-to-point transport to hub-and-spoke railway systems. Modern rail-rail transshipment yards accelerate container handling, so that multiple smaller trains with equal destination can be consolidated to a reduced number of trains without jeopardizing on time deliveries. An important problem continuously arising during the daily operations of a transshipment yard is the train scheduling problem, which decides on the succession of trains at the parallel railway tracks. This problem with a special focus on resolving deadlocks and avoiding multiple crane picks per container move is investigated within the paper on hand. A mathematical program along with a complexity proof is provided and exact (Dynamic Programming) and heuristic (Beam Search) procedures are described.

Keywords: Railway Systems; Transshipment Yard; Train Scheduling; Dynamic Programming; Beam Search

1 Introduction

With increasing volume of rail cargo an efficient operation of railway systems obviously becomes more and more important. In this context, modern rail-rail transshipment yards are an efficient alternative to traditional classification (or marshalling) yards, which both allow for a consolidation of different freight trains with equal destination. This way, hub-and-spoke systems, like they are traditionally applied for road transport, become available for railway systems, as well, and replace cost-intensive point-to-point transport of multiple small freight trains. Whereas traditional classification yards require a time-consuming reshuffling of railway cars via a system of track switches, in a modern transshipment yard container handling is conducted by huge gantry cranes spanning the railway tracks. Figure 1 depicts a schematic representation of a transshipment yard.

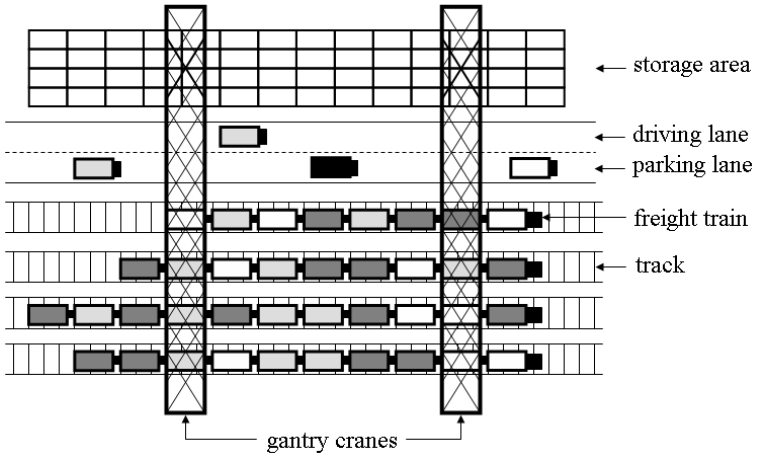

Figure 1: Schematic representation of a transshipment yard

A transshipment yard consists of a number G of parallel tracks on which different freight trains carrying multiple containers are processed. Container transshipment between freight trains is conducted by huge gantry cranes with a cantilever on both sides, which are arranged across the tracks. For instance, the largest German transshipment yard in Köln-Eifeltor consists of 9 parallel tracks and 6 successively arranged gantry cranes. Whereas parallel railway tracks and cranes are mandatory elements of a transshipment yard, truck lanes and a storage area are possible additional components:

(a) In a pure *rail-rail* system, which is exclusively dedicated to container transshipment between different freight trains, a storage area to intermediately stock containers amend the railway tracks. The storage area might also contain a sorting system, so that automated guided vehicles (Bostel and Dejax, 1998) or some rail-mounted vehicles (Alicke, 2002) are applied to pre-position containers. Such a pure rail-rail system labelled *Mega-Hub* is, for instance, located in Hannover-Lehrte (Germany) and is described in detail by Rotter (2004).

(b) A transshipment yard without a storage area, which is directly accessible by portal cranes, primarily serves as an interface between rail and road in a *rail-truck* intermodal system. These transhipment yards only consist of additional truck lanes (typically one lane for parking and another one for driving, see Ballis and Golias, 2002). A storage yard is often residing only near the yard and, thus, only accessible via additional truck moves. Consequently, a time-consuming rail-rail transshipment of containers in such a setting is only exceptionally applied. Such a system can be found, for instance, in Hamburg-Billwerder (Germany).

(c) Typically, transshipment yards have both a storage area and truck lanes, so that the yard serves as both a rail-rail transshipment yard and an interface for combined transport (see Ballis and Golias, 2002).

In what follows, we will especially treat yard types (a) and (c) and, thus, concentrate on rail-rail container transshipment. To allow for an efficient operation of such complex systems sophisticated and computerized scheduling procedures seem indispensable. However, the overall scheduling task seems far too complicated to allow for a simultaneous solution, so that a hierarchical decomposition of the overall problem is recommendable:

(i) Schedule the service slots of trains.

(ii) Assign each train to a railway track.

(iii) Decide on the containers' positions on trains.

(iv) Assign container moves to portal cranes.

(v) Decide on the sequence of container moves per crane.

The paper on hand exclusively treats problem (i) and provides scheduling procedures to assign trains to service slots. The problem can be characterized as follows: A given set I of trains, each of which is either already waiting on holding tracks near the yard or close to arrival, is to be assigned to different service slots $t = 1, \ldots, T$ of G simultaneously served trains. A transshipment yard is typically operated in distinct so-called pulses (Bostel and Dejax, 1998) or bundles (Alicke, 2002; Rotter, 2004) of trains, which means that G trains (one per track) are simultaneously served and jointly leave the system not before all container moves are processed, which are required for the respective bundle of trains. Then, another bundle of G trains enters the yard. Thus, for each train it is to be decided in which of the successive $t = 1, \ldots, T$ service slots of size G it is to be served. This assignment decision faces two important peculiarities:

Revisits: Each train stopping at the transshipment yard carries a given set of containers. A subset of these containers is already dedicated to the final destination of the train and, thus, remains untouched. The other part of containers is unloaded at the yard to be either stacked on other trains or to leave the rail system, e.g., by truck. Afterwards, vacant

railway cars can be filled with other containers (for the trains destination) either delivered from the storage yard or directly by another freight train. Thus, when deciding on the service slot of a train it is to be ensured that all containers to be loaded on this train have already been delivered by respective predecessor trains.

The property of trains being inbound and outbound vehicle simultaneously might result in deadlocks. Such a deadlock eventuates once all G tracks of a service slot are fully occupied by trains waiting for additional containers loaded on a train, which was not yet processed. Then, a deadlock can only be resolved by a subset of trains leaving the yard uncompleted, waiting on holding tracks until the respective train was processed and revisiting the yard in a later slot. Obviously, revisits delay shipment completion and are to be reduced to a minimum.

Split moves: Picking and dropping of containers by gantry cranes is precision work and, thus, very time-consuming (typically only 20-25 moves per crane and hour can be processed, see Rotter, 2004). If train i, which carries a container dedicated to another train j, is served in the same slot as train j, then the required container handling can be conducted by a single crane move directly from train i to j. On the other hand, an assignment of train i to a prior slot than train j requires that the container is intermediately stocked in the storage yard. The container movement is interrupted and two separated and time-consuming crane picks and drops are required, which we label as a split move. Hence, train scheduling aims at avoiding split moves to accelerate container handling.

The paper on hand provides a train scheduling procedure which covers both peculiarities. For this purpose, the remainder of the paper is structured as follows. Section 2 summarizes existing literature on transshipment yards. Then, Section 3 describes the train scheduling problem in detail, formalizes the problem as a mathematical program and states complexity. Section 4 presents exact (Dynamic Programming) and heuristic (myopic start procedure and Beam Search) solution procedures, which are evaluated in a comprehensive computational study (Section 5). Finally, Section 6 summarizes the paper and specifies future research challenges.

2 Literature review

Although there is a lot of attention paid to railway optimization (see, e.g., Cordeau et al. 1998) and intermodal transportation (see Crainic and Kim, 2007) in general, literature on rail-rail transshipment yards is scarce. Current research on transshipment in rail systems mainly focuses on traditional shunting yards. In these conventional yards, scheduling trains and rearranging freight trains via shunting hills and a system of track switches is covered, e.g., by Blasum et al. (2000), Dahlhaus et al. (2000), He et al. (2000), Winter and Zimmermann (2000) and Freling et al. (2005). However, modern rail-rail transshipment yards, where container transshipment is conducted by gantry cranes without rearranging the railway cars by themselves (see Section 1), are an emerging technology in rail systems

(see the surveys by Bontekoning et al., 2004 as well as Macharis and Bontekoning, 2004) and promise a more efficient transshipment.

In-depth descriptions of structural properties and different operational policies employed in rail-rail transshipment yards are provided by Ballis and Golias (2002) as well as Rotter (2004). Meyer (1998) and Wiegmans et al. (2006) specifically address the design process of an optimal terminal layout. However, only very few research papers deal with the scheduling problems perpetually arising during the daily operations of a transshipment yards. According to the hierarchical decomposition scheme of the overall scheduling problem (see Section 1) Bostel and Dejax (1998) as well as Corry and Kozan (2007) treat problem (iii) and provide scheduling procedures to determine the optimal positions of containers on inbound and outbound trains so that crane moves at the yard are minimized. Alicke (2002) provides a scheduling procedure to jointly cover problems (iv) and (v). Based on constraint programming Alicke assigns container moves to cranes, which are not allowed to interfere, and decides on the sequence of moves per crane. Related problems also occur within container terminals (see, e.g., Zhu and Lim, 2006; Moccia et al., 2006; Lim et al., 2007; Sammarra et al., 2007). Finally, problem (ii), the assignment of trains to tracks, can be solved as a quadratic assignment problem, which is shown by Alicke and Arnold (1998), if the schedule of trains (problem (i)) was previously solved.

Up to now, there exists no literature related to problem (i), which is in the focus of the paper on hand. In what follows a detailed problem statement of how to schedule the arrival and processing of trains at the available service slots is provided.

3 Problem statement

3.1 Mathematical program

The basic decision of the transshipment yard scheduling problem (TYSP) is to assign each train i of a given train set I to a service slot $t = 1, \ldots, T$. To each slot t at most G trains can be assigned, because G is the number of parallel railway tracks of the transshipment yard. Note that an assignment of trains to dedicated tracks is not part of TYSP (see Section 2), so that the assignment decision can be recorded by binary variables x_{it}, which receives the value one, whenever train i is assigned to slot t (0, otherwise). The overall number T of slots required to process the given set I of trains amounts to: $T = \lceil \frac{|I|}{G} \rceil$. However to ease the description, w.l.o.g. we assume that: $|I| = G \cdot T$, which can be easily ensured for any problem instance by inserting empty trains carrying no containers. Furthermore we assume that no assignment restrictions between trains and slots exist. Required alterations, when some trains arrive only after the first slots have already passed or need to be assigned to earlier slots to meet timetable requirements, are discussed in Section 7.

This assignment decision aims at realizing two (partially conflicting) objectives: The number of revisits of trains (objective 1) and the number of split moves of containers (objective 2) are to be minimized. To operationalize the simultaneous optimization of both objectives we assume that a linear combinations of the single objectives is to be minimized

I	set of trains (indices i and j)
L_i	set trains carrying containers dedicated to train i
T	number of time slots for (un-)loading trains (index t)
G	number of parallel tracks within transshipment yard
A_{ij}	number of containers train i receives from train j
M	big integer (e.g., $M = T - 1$)
α_1, α_2	given weights for objectives 1 and 2 with α_1, $\alpha_2 \geq 0$
x_{it}	binary variable: 1, if train i is assigned to slot t; 0, otherwise
y_i	binary variable: 1, if train i has to revisit the yard; 0, otherwise
z_{ij}	binary variable: 1, if trains i and j are served during different slots; 0, otherwise

Table 1: Notation

and weights α_1 and α_2 can be assigned to each objective. For instance, whenever a revisit of a train delays an overall transshipment schedule more than a few additional split moves, it might be a reasonable choice to bring both objectives in lexicographic order by choosing the weights as follows: $\alpha_1 = \sum_{i \in I} \sum_{j \in L_i} A_{ij}$ and $\alpha_2 = 1$, where L_i is the set of trains a train i receives a number A_{ij} of containers from. Another possibility is to derive a set of Pareto optimal solutions using representative data. Based on the obtained Pareto frontier and the experience of yard managers weights can be chosen in the most suitable way. In the following, we presuppose that suited weights already exist.

With the help of the notation summarized in Table 1 TYSP consists of objective function (1) and constraints (2) to (6):

$$\text{(TYSP) Minimize } C(X, Y, Z) = \alpha_1 \sum_{i \in I} y_i + \alpha_2 \sum_{i \in I} \sum_{j \in L_i} z_{ij} \cdot A_{ij} \tag{1}$$

subject to

$$\sum_{t=1}^{T} x_{it} = 1 \qquad \forall\, i \in I \tag{2}$$

$$\sum_{i \in I} x_{it} \leq G \qquad \forall\, t = 1, \ldots, T \tag{3}$$

$$\sum_{t=1}^{T} x_{it} \cdot t + y_i \cdot M \geq \sum_{t=1}^{T} x_{jt} \cdot t \qquad \forall\, i \in I;\, j \in L_i \tag{4}$$

$$\left| \sum_{t=1}^{T} x_{it} \cdot t - \sum_{t=1}^{T} x_{jt} \cdot t \right| \leq z_{ij} \cdot M \qquad \forall\, i \in I;\, j \in L_i \tag{5}$$

$$x_{it};\, y_i;\, z_{ij} \in \{0, 1\} \qquad \forall\, i \in I;\, t = 1, \ldots, T;\, j \in L_i \tag{6}$$

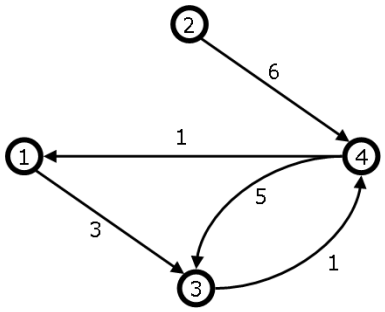

Figure 2: Example data

Objective function (1) minimizes a linear combination of objectives 1 (number of revisits) and 2 (number of split moves) weighted by α_1 and α_2, respectively. The number of revisits is calculated by summing binary variables y_i, which record whether train i has to revisit ($y_i = 1$) or not ($y_i = 0$). The calculation of split moves relies on the consideration, that once a train j carries a number A_{ij} of containers dedicated to train i all of these A_{ij} containers must be picked and dropped twice and, thus, require split moves, whenever trains i and j are not assigned to the same slot ($z_{ij} = 1$). Equalities (2) ensure that each train is assigned to exactly one slot, whereas constraints (3) force the number of trains per slot to not exceed the given number G of tracks. Constraints (4) force binary variable y_i to receive the value one, whenever a train j, which carries containers dedicated to train i, is assigned to a later slot than train i. In a similar fashion, constraints (5) ensure, that binary variables z_{ij} receive the value one, whenever trains i and j are assigned to different slots (0, otherwise). Big integer M, applied in both constraints (4) and (5), can be set to $T-1$. Note that constraints (5) can be easily linearized, however, for means of conciseness we abstain from a detailed description.

Example: The input data for a TYSP-instance can be represented by a digraph $G = (I, E, A)$. A vertex is introduced for each train $i \in I$. An arc from vertex (train) j to i indicates that train j carries containers dedicated to train i, so that j belongs to the set L_i of predecessor trains of train i. Finally, the number A_{ij} of containers to be moved from j to i are used as arc weights. Figure 2 displays such an input graph for an example with 4 trains, which are to be assigned to $T = 2$ slots at a transshipment yard with $G = 2$ tracks.

Consider a solution where trains 2 and 3 are assigned to the first and trains 1 and 4 to the second slot. Then, train 3 has to revisit because the 3 and 5 containers to be received from trains 1 and 4, respectively, have not yet arrived. Moreover, 15 split moves have to be processed because out of overall 16 container moves required only the single container transshipped from train 4 to 1 can be directly processed. Consequently, the overall solution value amounts to $C = \alpha_1 + 15\alpha_2$.

3.2 Complexity

In the following we will proof NP-completeness for $TYSP$ by a reduction from *Acyclic Partition* which is well known to be NP-complete in the strong sense (see Garey and Johnson, 1979) and can be summarized as follows:

Acylic Partition: An instance of the Acyclic Partition problem consists of a digraph $\Gamma = (V, A)$ with vertex set V and arc set A. Additionally given are two positive integers B and K as well as weights $w(v) \in \mathbb{N}$ for each vertex $v \in V$. Finally, to each arc $a \in A$ costs $c(a) \in \mathbb{N}$ are assigned.
Is there a partition of V into disjoint sets $V_1, V_2, ..., V_m$ such that the derived directed graph $\Gamma' = (V', A')$ is acyclic, where a vertex is inserted for any subset of nodes: $V' = V_1, ..., V_m$, and an arc (V_i, V_j) whenever there exists an arc $(v_i, v_j) \in A$ for some pair of vertices $v_i \in V_i$ and $v_j \in V_j$? Furthermore it must hold that (i) the sum of vertex weights in each subset V_i does not exceed B and (ii) the sum of the costs of all those arcs having their defining vertices in different sets does not exceed K.

Comment: The problem remains NP-complete even if all vertex weights and all arc weights of Γ are equal to 1.

Proof: TYSP is obviously in NP. Furthermore, we can polynomially transform Acyclic Partition to TYSP, which is shown in the following. For a given instance $I(AP)$ of the Acyclic Partition problem with $w(v) = 1 \,\forall\, v \in V$ we construct an instance $I(T)$ of TYSP: Define $G := B; \alpha_1 := K + 1; \alpha_2 := 1$ and $T := m$. The set of trains I corresponds to the vertex set V of graph Γ and for each arc $a \in A$ connecting vertices v_i and v_j the number A_{ij} of containers (delivered from train i to train j) corresponds to the arc cost, i.e., $A_{ij} := c(a)$. We will show that $I(AP)$ has a solution if and only if there exists a train schedule for $I(T)$ with an objective value not exceeding K.

If $I(AP)$ has a solution, vertex set V can be partitioned into disjoint subsets $V' = V_1, ..., V_m$ such that $\mid V_i \mid \leq B = G \,\forall\, i = 1, ..., m$ and $\Gamma' = (V', A')$ is acyclic. Γ' may be topologically ordered where all trains represented by vertices in V_i are assigned to time slot $t = i$ and any arc connecting V_j to V_i in Γ' implies $i > j$. Thus, there are no revisits and $y_i = 0 \,\forall\, i \in I$. Let (V_i, V_j) be an arc in Γ' then $z_{ij} = 1 \,\forall\, (v_i, v_j) \in A$. Hence the objective function value of $I(T)$ is $0 + \sum_i \sum_j z_{ij} \cdot A_{ij} = \sum_{\{a \in A | a = (v_i, v_j), v_i \in V_i, v_j \in V_j : i \neq j\}} c(a) \leq K$.

Consider a train schedule for $I(T)$ with objective of at most K. As an immediate consequence all variables $y_i = 0$ and $\sum_{i \in I} \sum_{j \in L_i} z_{ij} \cdot A_{ij} \leq K$. Let $V_1, ..., V_m$ be the number of vertices corresponding to trains assigned to time slots $t = 1, ..., T = m$. Obviously every V_i contains at most B elements and the total number of containers carried between trains assigned to different time slots equals the total weight of arcs connecting vertices from different sets V_i which is at most K. As there are no revisits $I(AP)$ has a solution.

Note that a special case of TYSP for an acyclic directed input graph can be solved in polynomial time. Under these prerequisites, the problem reduces to a topological ordering

of vertices, which is solvable in $O(|I|^2)$ time (see Lawler, 1976, p. 30). The slots can simply be filled in chronological order according to the obtained numbering of vertices.

4 Algorithms

In this section three solution procedures for TYSP are presented: an exact Dynamic Programming approach (Section 4.1), a myopic start procedure (Section 4.2) and a Beam Search procedure.

4.1 An exact Dynamic Programming approach

The Dynamic Programming (DP) approach is based on an acyclic digraph $H = (V, E, r)$ with a vertex set V divided into $T+1$ *stages*, a set E of arcs connecting vertices of adjacent stages and an arc weighting function $r : E \to \mathbb{R}$ (see Boysen et al., 2007, for a related approach to sequencing mixed-model assembly lines). Each service slot t is represented by a stage which contains a subset $V_t \subset V$ of vertices representing *states* of the partial train schedule up to slot t. The first stage, $t = 0$, represents the empty set. Each vertex $j \in V_t$ identifies a state (t, j) defined by a set I_{tj}, which contains all freight trains $i \in I$ already scheduled up to service slot t. It is sufficient to store the scheduled trains instead of their exact partial slot assignment, because the contribution to the overall objective value of a train set actually served at service slot $t + 1$ only depends on the trains scheduled before (stored in I_{tj}) irrespective of their exact order.

The following conditions define all *feasible* states to be represented as vertices of the graph:

$$|I_{tj}| = t \cdot G \qquad \forall\, t = 0, \ldots, T;\ j \in V_t \tag{7}$$

$$I_{tj} \subseteq I \qquad \forall\, t = 0, \ldots, T;\ j \in V_t \tag{8}$$

Obviously, the vertex set V_0 contains only a single vertex (initial state $(0, 1)$) corresponding to the train set $I_{01} = \emptyset$. Similarly, vertex set V_T contains a single vertx (final state $(T, 1)$) with $I_{T1} = I$. The remaining stages have a variable number of vertices depending on the number of different train sets I_{tj} possible.

Two vertices (t, j) and $(t + 1, k)$ of two consecutive stages t and $t + 1$ are connected by an arc if the associated train sets I_{tj} and I_{t+1k} differ only in G elements, i.e., exactly G freight trains are additionally scheduled in position $t + 1$. This is true if $I_{tj} \subset I_{t+1k}$ holds, because both states are feasible according to (7) and (8). By I_{tjk} we denote the set of trains associated with the arc between a vertex pair (t, j) and $(t + 1, k)$. The overall arc set is defined as follows:

$$E = \{((t, j), (t + 1, k)) \mid t = 0, \ldots, T - 1; j \in V_t; k \in V_{t+1} : I_{tj} \subset I_{t+1k}\} \tag{9}$$

Finally, arc weights $r_{((t,j),(t+1,k))}$ assign the contribution to the overall objective value to each arc (and the scheduling decision represented by the arc). This weight is a linear

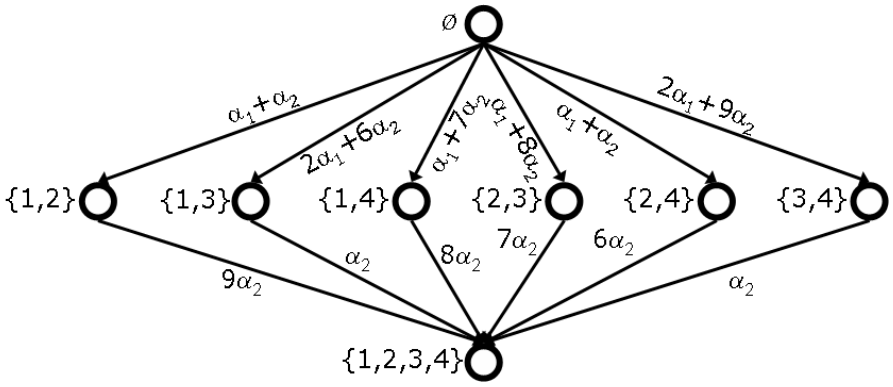

Figure 3: Example graph for Dynamic Programming

combination of minimizing the number of revisits (first term) and of minimizing the number of split moves (second term) and is calculated as follows:

$$r_{((t,j),(t+1,k))} = \alpha_1 \cdot \sum_{i \in I_{tjk}} \min\{1;\, |L_i \setminus I_{t+1k}|\} + \alpha_2 \cdot \sum_{i \in I_{tjk}} \sum_{\tau \in \{L_i \setminus I_{tjk}\}} A_{i\tau} \tag{10}$$

Any train $i \in I_{tjk}$ processed during slots $t+1$ (and, thus, associated with the respective arc) has to revisit, when any of its predecessors L_i has not been assigned up to actual slot $t+1$ (and is, thus, not in set I_{t+1k}), so that the relative complement of I_{t+1k} in L_i is not the empty set. The minimum function ensures that only a single revisit is required even if more than one predecessor train has not yet been scheduled. Furthermore, all containers $A_{i\tau}$ dedicated to a actually scheduled train $i \in I_{tjk}$ to be delivered by predecessor train $\tau \in L_i$ are to be processed by a split move, if these two trains are not jointly served in the actual slot $t+1$.

With this graph on hand, the optimal solution of our train scheduling problem reduces to finding the shortest path from the unique source vertex at level 0 to the unique sink vertex at level T, where the length of the path is given by the sum of weights of the arcs contained. The minimum objective value corresponds to the length of the shortest path. The G freight trains to be assigned to service slot $t+1$ equal those contained in train set I_{tjk} associated with any arc $((t,j),(t+1,k))$ for $t = 0, ..., T-1$ on the shortest path SP.

Example (cont.): The resulting DP graph for our example is depicted in Figure 3. The optimal solution is to assign trains 2 and 4 to the first slot and trains 1 and 3 to the second. Note that in this example the optimal solution does not depend on the choice of objective weights α_1 and α_2. The resulting objective value is $C = \alpha_1 + 7\alpha_2$.

Instead of constructing the complete graph before computing the shortest path, the more efficient DP approach consists of determining the shortest path from the initial state to each vertex stage-by-stage ($t = 0, ..., T-1$). In order to do so, only two stages of the graph have to be stored simultaneously, because the shortest path to a vertex $(t+1, j)$ in

stage $t+1$ is composed of a shortest path to a vertex (t,i) in stage t (already determined and stored) and the connecting arc $((t,i),(t+1,j))$. Among all such paths to $(t+1,j)$ one with minimal sum of arc weights (length of path to (t,i) plus $r_{((t,i),(t+1,j))}$) is to be selected. The length-minimizing vertex (t,i) is stored as the predecessor in the shortest path to $(t+1,j)$ together with the length of this path. After reaching the final state $(T,1)$ in stage T, the optimal path can be retrieved in backward direction stage-by-stage using the stored predecessor vertices.

4.2 A myopic start procedure

As the problem was shown to be NP-complete in the strong sense, heuristic solution approaches are required to solve problem instances of real-world size. First, a simple myopic start procedure (MSP) is presented, which simply fills available time slots $t = 1,\ldots,T$ and $g = 1,\ldots,G$ tracks in increasing order until all trains are scheduled. The actual choice at a decision point is the train $i \in POS$ (where POS is the set of trains not yet scheduled), which increases the objective value least. A formal description of MSP is as follows:

(0) Initialize the following data: $POS := I$; $t := 1$.

(1) Initialize train set IN_t of trains already scheduled within time slot t: $IN_t := \emptyset$ and track counter: $g := 1$.

(2) Determine the actual train *sel* to be assigned at the actual decision point:

$$sel := \text{argmin}_{i \in POS} \left\{ \alpha_1 \cdot \min\{1; |L_i \setminus \{I \setminus POS\}|\} + \alpha_2 \cdot \sum_{\tau \in \{L_i \setminus IN_t\}} A_{i\tau} \right\} \qquad (11)$$

(3) Update the trains sets: $IN_t := IN_t \cup \{sel\}$ and $POS := POS \setminus \{sel\}$.

(4) Increase track counter: $g := g+1$ and decide on the advancement of the procedure: If all tracks of the actual time slot are filled with trains ($g > G$), then increase the time slot: $t := t+1$ and proceed with step (1) (or goto final step (5) if all time slots are filled with trains ($t > T$)), else goto step (2).

(5) Determine the objective value C of the solution stored in sets IN_t.

Example (cont.): Table 2 displays the successive train assignment of MSP for our example, resulting in a train schedule where trains 1 and 2 are served in time slot 1 and trains 3 and 4 in slot 2. Note that the minimum train number is applied to break ties. In this example, the solution process is independent of the objective weights α_1 and α_2 and the overall objective value amounts to $C = \alpha_1 + 10\alpha_2$.

$POS(t=1;g=1)$	$POS(t=1;g=2)$	$POS(t=2;g=1)$	$POS(t=2;g=2)$
$1 \rightarrow \alpha_1 + \alpha_2$	$\underline{1 \rightarrow \alpha_1 + \alpha_2}$	$3 \rightarrow \alpha_1 + 5\alpha_2$	$\underline{3 \rightarrow 0}$
$\underline{2 \rightarrow 0}$	$3 \rightarrow \alpha_1 + 8\alpha_2$	$\underline{4 \rightarrow \alpha_1 + \alpha_2}$	
$3 \rightarrow \alpha_1 + 8\alpha_2$	$4 \rightarrow \alpha_1 + \alpha_2$		
$4 \rightarrow \alpha_1 + 7\alpha_2$			

Table 2: Solution process of MSP

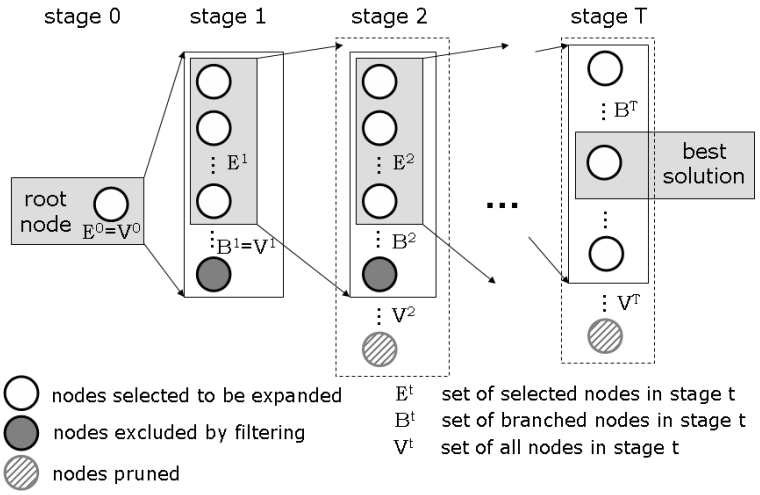

Figure 4: Schematic representation of Beam Search

4.3 A heuristic Beam Search approach

4.3.1 General description

Beam Search is a truncated breadth-first search heuristic and was first applied to speech recognition systems by Lowerre (1976). Ow and Morton (1988) were the first to systematically study the performance of Beam Search compared to other well-known heuristics for two scheduling problems. Since then, Beam Search was applied within multiple fields of application and many extensions have been developed, e.g., stochastic vertex choice (Wang and Lim, 2007) or hybridization with other meta-heuristics (Blum, 2005), so that Beam Search turns out to be a powerful meta-heuristic applicable to many real-world optimization problems. A review on these developments is provided by Sabuncuoglu et al. (2008).

Like other breadth-first search procedures Beam Search bases on a tree representation of the solution process. However, unlike a breadth-first version of Branch&Bound, Beam Search also excludes vertices, which might contain optimal solutions, and restricts the number of vertices per stage to be further branched to a promising subset. This subset of nodes is determined by heuristic choice in a filtering process. A schematic representation of Beam Search is depicted in Figure 4.

Starting with the root vertex of stage 0, all vertices (set V^1) of stage 1 are constructed

and form the set B^1 of branched vertices. Then, the filtering process of Beam Search starts to identify promising vertices of stage 1. Such a measure can, for instance, be obtained by a simple priority value, a lower bound procedure or even an upper bound, solutions constructed by completing partial solutions, e.g., with a simple myopic priority rule based heuristic (see Ow and Morton, 1988). Moreover, a multi-stage filtering procedure can be applied, where filtering procedures are ordered from rough to detailed filtering (see Sabuncuoglu et al., 2008). By applying one of these alternatives a priority value is assigned to each vertex within set B^1. With regard to this priority value the first BW vertices are chosen to form the set E^1 of size $|E^1| = BW$, where BW is a control parameter called *beam width*. Only the selected vertices contained in set E^1 are expended and branched to build the set B^2 of branched vertices in stage 2, which is only a subset of all possible vertices V^2. These steps are repeated until the final stage T is reached, where the best solution out of the set B^T of constructed vertices is returned as the result of the Beam Search procedure.

4.3.2 A Beam Search procedure for TYSP

To apply the general procedure of Beam Search in a specific domain the following two components must be specified with regard to the respective problem: (i) graph structure and (ii) filtering. In the following, we describe these specifications for TYSP:

Graph structure: As graph structure required for a Beam Search application to TYSP, the acyclic digraph $H = (V, E, r)$ already specified for the Dynamic Programming procedure (Section 4.1) can directly be utilized.

Filtering: To select a number of BW (beam width) vertices out of the set B^t into vertex set E^t per stage t, we apply the most simple filtering process, which is to simply calculate the actual objective value of the respective partial solution. This value amounts to the sum of arc weights along the shortest path leading to the actual vertex. Note that Beam Search does not construct the complete graph as specified in Section 4.1 but only a subgraph, so that merely the shortest path within the resulting subgraph is employed. We abstain from more sophisticated filtering procedures as these procedures require additional computational effort and our computational study of Section 5 reveals a promising solution quality of basic Beam Search.

With these domain specific choices on hand, our Beam Search approach stage-wise identifies a subset of promising vertices. Finally, when the last stage is reached the best solution value is returned as the solution of the Beam Search approach.

5 Computational study

Thus far, there is no established test-bed available for the train scheduling problem treated in this paper. Therefore, we first elaborate on the instances that are used in our computational study. Then, numerical results on the performance of the proposed algorithms are discussed.

5.1 Instance generation

To derive test instances the input parameters listed in Table 3 are used to produce the sets L_i of trains carrying containers dedicated to train i and the number A_{ij} of containers to be moved between trains i and j defining a TYSP instance. We differentiate between case A (6-15 trains on 3 tracks) and case B instances (24-36 trains on 4 tracks), where the size of case A instances is reduced so that all instances can be solved to optimality.

symbol	description	case A	case B
$\|I\|$	number of trains	6, 9, 12, 15	24, 28, 32, 36
G	number of tracks	3	4
$Prob$	probability of train j carrying containers for trains i	0.2, 0.4, 0.6, 0.8	

Table 3: Parameters for instance generation

In both test cases, these parameters are combined in a full-factorial design and in each parameter constellation instance generation is repeated 20 times, so that $2 \cdot 4 \cdot 4 \cdot 20 = 640$ different instances were obtained. On the basis of a given set of parameters each single instance is generated as follows: For each single possible relationship between trains $i \in I$ and $j \in I$ according to parameter $Prob$ it is randomly drawn whether train j carries containers dedicated to train i or not. If so, the number A_{ij} of containers to be transshipped is randomly drawn (with equal distribution) out of the interval $[1, 10]$. Finally, both objective weights are set to one: $\alpha_1 = \alpha_2 = 1$.

5.2 Computational results

All methods have been implemented in C# (Visual Studio 2003) and run on a Pentium IV, 1800 MHz PC, with 512 MB of memory. First, the performance of Beam Search (BS) in dependency of control parameter BW is evaluated. Figure 5 displays solution performance and solution time, where solution performance is represented by the relative deviation from optimum averaged over all case A instances (labeled *avg gap* and measured by $\frac{C(BS)-C(DP)}{C(DP)} \cdot 100$, where $C(BS)$ and $C(DP)$ are the objective values of heuristic Beam Search and exact Dynamic Programming, respectively). Solution time is measured by the average CPU-seconds per instance (labeled *avg cpu*).

Figure 5 depicts a linear increase of solution time with increasing beam width BW. This result is not astounding as the number of vertices to be inspected in the graph increases linear with BW. On the other hand, solution quality increases (indicated by an decrease of *avg gap*) subproportionally with increasing BW. This result is also plausible, because the probability of vertices being part of a good overall solution more and more decreases the farther a vertex's shortest path deviates from the best value of the respective stage.

In a second experiment, performance of all procedures is compared for case A instances. All of these instances can be solved to optimality with the Dynamic Programming (DP) approach, so that the solution performance of our myopic start procedure (MSP) and Beam

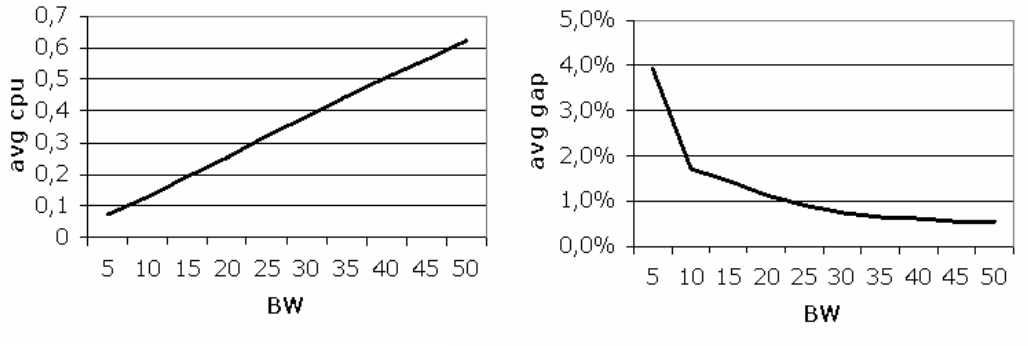

Figure 5: Solution time and solution quality of Beam Search with varying beam width BW

Search (BS) can be compared in relation to optimal solutions. BS is executed with a given control parameter of $BW = 30$. Table 4 lists the results of this comparison. In addition to the aforementioned quality measures, *max gap* is reported, which denotes the maximum relative deviation from the optimal solution over all instances per parameter constellation.

With regard to DP it can be stated, that solution time increases exponentially, so that no instance with $|I| = 18$ trains could be solved to optimality within a given time frame of 300 CPU-seconds. On the other hand, MSP requires negligible solution time of less than 0.1 CPU-seconds per instance, but produces a considerable gap compared to optimal solutions, which is the broader the less containers are to be moved between trains (indicated by lower *Prob*). This coherency results from the fact that any misled myopic choice preponderates if the overall objective value is on a lower level. The same finding also holds for BS, but with a much smaller deviation from optimum. BS results in a total *avg gap* of merely 0.7% with a average solution time of only 0.4 CPU-seconds, so that near optimal solutions can be obtained in a very short time frame.

In a last experiment, the heuristic procedures are compared for case B instances, which cannot be solved to optimality. Instead, we compare BS, which is executed with a control parameter $BW = 5$, and MSP with a first-come-first-serve policy (FCFS), which is often applied in real-world transshipment yards to solve the train scheduling problem. The results of FCFS are emulated for our test instances by assigning trains to tracks and slots according to increasing train number. Consequently, average and maximum gaps reported in Table 5 rely on deviations per instance calculated as follows: $\frac{C(FCFS)-C(x)}{C(FCFS)} \cdot 100 \, \forall \, x \in \{MSP, BS\}$.

With regard to solution quality BS clearly outperforms the other procedures, MSP and FCFS. BS increases the FCFS solution by 41.5% on average and results in a superior objective value for each single instance. However, solution time increases considerably to 376.6 CPU-seconds on average for $|I| = 36$ trains, so that MSP might be a reasonable choice for larger instances. Here solution time exceeds 0.1 CPU-seconds in neither instance and FCFS-solution are still improved by 14.9% on average up to a maximum of 60%.

		DP	MSP		BS		
$\|I\|$	*Prob*	avg cpu	avg gap	max gap	avg gap	max gap	avg cpu
6	0.2	<0.01	62.5	133.3	0.0	0.0	<0.1
	0.4	<0.01	46.8	93.3	0.0	0.0	<0.1
	0.6	<0.01	31.9	85.7	0.0	0.0	<0.1
	0.8	<0.01	22.4	45.3	0.0	0.0	<0.1
9	0.2	0.03	63.9	100.0	0.0	0.0	<0.1
	0.4	0.06	43.6	83.7	0.1	2.3	<0.1
	0.6	0.07	31.9	57.6	0.4	3.4	<0.1
	0.8	0.08	21.2	39.1	0.0	0.7	<0.1
12	0.2	1.38	71.3	100.0	0.8	16.7	0.3
	0.4	2.20	49.5	80.2	2.2	5.6	0.3
	0.6	2.30	26.8	40.0	0.6	1.4	0.3
	0.8	2.41	18.6	26.4	0.4	2.3	0.3
15	0.2	61.99	72.7	96.7	2.7	17.6	1.1
	0.4	73.41	44.4	69.1	2.4	8.9	1.2
	0.6	73.54	27.7	35.5	1.4	3.3	1.2
	0.8	75.54	18.2	25.9	0.9	2.8	1.7
	total	18.31	39.8	133.3	0.7	17.6	0.4

Table 4: Performance of procedures for case A instances

		MSP		BS		
$\|I\|$	*Prob*	avg gap	max gap	avg gap	max gap	avg cpu
24	0.2	36.9	53.2	79.8	91.7	15.0
	0.4	14.6	32.3	47.4	72.5	15.0
	0.6	10.3	17.0	29.5	34.8	15.4
	0.8	5.2	8.8	18,2	21.7	16.0
28	0.2	35.1	60.0	80.3	92.9	49.4
	0.4	15.2	28.9	45.5	53.5	50.1
	0.6	8.4	12.4	28.0	31.2	51.0
	0.8	5.0	7.8	18.2	20.7	51.4
32	0.2	30.1	46.6	78.2	91.4	140.9
	0.4	13.9	22.2	41.9	51.2	142.1
	0.6	7.2	11.4	25.1	29.1	146.7
	0.8	3.9	8.4	16.6	19.5	145.8
36	0.2	30.6	43.1	74.6	87.2	378.1
	0.4	11.9	21.1	39.9	48.6	372.4
	0.6	7.2	10.5	24.9	28.2	377.7
	0.8	3.5	6.0	16.1	19.0	378.1
	total	14.9	60.0	41.5	92.9	146.6

Table 5: Performance of procedures for case B instances

6 Conclusions

This paper investigates a novel train scheduling problem, which arises during container transshipment at modern rail-rail transshipment yards. Trains need to be unified to jointly processed bundles of G trains (each track receives a single train), which are to be assigned to service slots. This assignment decision aims at avoiding revisits of trains, which are inevitable whenever a train is processed in a slot prior to the arrival of all containers dedicated to this train, and the minimization of split moves, which require multiple time-consuming crane picks and drops for a single container move between two trains. Different exact (Dynamic Programming) and heuristic (myopic start procedure and Beam Search) solution procedures are presented. The computational tests indicate the appropriateness of the developed procedures to considerably improve a first-come-first-serve policy, which is a traditional choice in real-world transshipment yards. However, there remains some future research to be done.

Typically, trains are bound to time tables, which restrict the slots to which a train is assignable. A train must not be assigned to a slot ahead of its arrival. Just as well, trains are to be supplied with all its required containers, so that an on time departure according to the time table of the superordinate railway system is assured. In this case, TYSP has to consider time table requirements by incorporating earliest e_i and latest l_i service slots per train i:

$$e_i \leq \sum_{t=1}^{T} x_{it} \cdot t \leq l_i \quad \forall\, i \in I \tag{12}$$

Note that within TYSP the duration of slots is not explicitly calculated, so that historical data is to be applied to estimate a representative time span. These earliest and latest processing times can be easily incorporated into the graph structure, DP and Beam Search rely on. As an arc captures the processing of a dedicated bundle of trains in a specific service slot, those arcs which lead to untimely train assignments just need to be excluded from the graph. Consequently, the graph structure is reduced with the eligible side effect of accelerating the solution process.

Additionally, time tables might also be bound to changes like canceled or additional trains. In this case, TYSP becomes an online problem, where the set of trains to be scheduled continuously changes during processing a once derived plan. If an already scheduled train fails to appear or additional unplanned trains arrive these changes need to be considered appropriately in a rolling planning horizon. However, in the real-world TYSP is often indeed a static problem. Because of their lower speed, in many European countries, freight trains are only allowed to travel overnight. As a result of this policy, commercial trains, typically, arrive at a transshipment yard in the early morning, are served during the day and leave the yard for their final destinations the other night (see Ballis and Golias, 2002). Thus, TYSP predominantly faces a static set of trains, which is to be processed during the day.

Finally, an appropriate integration of related scheduling problems (see Section 1), i.e.,

track assignment and crane scheduling, which are highly interdependent with train scheduling, in a hierarchical planning approach is another challenging task for future research.

References

[1] Alicke, K., 2002. Modeling and optimization of the intermodal terminal Mega Hub, OR Spectrum 24, 1–17.

[2] Alicke, K., Arnold, D., 1998. Optimierung von mehrstufigen Umschlagsystemen, Fördern und Heben 8, 769–772.

[3] Ballis, A., Golias, J., 2002. Comparative evaluation of existing and innovative rail-road freight transport terminals, Transportation Research Part A 26, 593–611.

[4] Blasum, U., Bussieck, M.R., Hochstättler, W., Moll, C., Scheel, H.-H., Winter, T., 2000. Scheduling trams in the morning, Mathematical Methods of Operations Research 49, 137–148.

[5] Blum, C., 2005. Beam-ACO – Hybridizing ant colony optimization with beam serach: An application to open shop scheduling, Computers & Operations Research 32, 1565–1591.

[6] Bontekoning, Y.M., Macharis, C, Trip, J.J., 2004. Is a new applied transportation research flied emerging? A review of intermodal rail-truck freight transport literature, Transportation Research Part A 38, 1–24.

[7] Bostel, N., Dejax, P., 1998. Models and algorithms for container allocation problems on trains in a rapid transshipment shunting yard, Transportation Science 32, 370–379.

[8] Boysen, N., Fliedner, M., Scholl, A., 2007. Sequencing mixed-model assembly lines to minimize part inventory cost, OR Spectrum (to appear).

[9] Cordeau, J.-F., Toth, P., Vigo, D., 1998. A survey of optimization models for train routing and scheduling, Transportation Science 32, 380–404.

[10] Corry, P., Kozan, E., 2007. Optimised loading patterns for intermodal trains, OR Spectrum (to appear).

[11] Crainic, T.G., Kim, K.H., 2007. Intermodal transport, In: Barnhart, C., Laporte, G. (eds.) Transportaion, Handbooks in Operations Research and Management Science 14, 467–538.

[12] Dahlhaus, E., Horak, P., Miller, M., Ryan, J.F., 2000. The train marshalling problem, Discrete Applied Mathematics 103, 41–54.

[13] Freling, R., Lentink, R.M., Kroon, L.G., Huisman, D., 2005. Shunting of passenger train units in a railway station, Transportation Science 39, 261–272.

[14] Garey, M.R., Johnson, D.S., 1979. Computers and intractability: A guide to the theory of NP-completeness, Freeman, New York.

[15] He, S., Song, R., Chaudhry, S.S., 2000. Fuzzy dispatching model and genetic algorithms for railyards operations, European Journal of Operational Research 124, 307–331.

[16] Lawler, E., 1976. Combinatorial optimization: Networks and matroids, New York.

[17] Lim, A., Rodrigues, B., Xu, Z., 2007. A m-parallel crane scheduling problem with a non-crossing constraint, Naval Research Logistics 54, 115–235.

[18] Lowerre, B.T., 1976. The HARPY speech recognition system, Ph.D. thesis, Carnegie-Mellon University, U.S.A., April.

[19] Macharis, C., Bontekoning, Y.M., 2004. Opportunities for OR in intermodal freight transport research: A review, European Journal of Operational Research 153, 400–416.

[20] Meyer, P., 1998. Entwicklung eines Simulationsprogramms für Umschlagterminals des Kombinierten Verkehrs, Ph.D. thesis, Universität Hannover.

[21] Moccia, L., Cordeau, J.F., Gaudioso, M., Laporte, G., 2006. A branch-and-cut algorithm for the quay crane scheduling problem in a container terminal, Naval Research Logistics 53, 45–59.

[22] Ow, P.S., Morton, T.E., 1988. Filtered beam search in scheduling, International Journal of Production Research 26, 297–307.

[23] Rotter, H., 2004. New operating concepts for intermodal transport: The Mega Hub in Hanover/Lehrte in Germany, Transportation Planning and Technology 27, 347–365.

[24] Sabuncuoglu, I., Gocgun, Y., Erel, E., 2008. Backtracking and exchange of information: Methods to enhance a beam search algorithm for assembly line scheduling, European Journal of Operational Research 186, 915–930.

[25] Sammarra, M., Cordeau, J.F., Laporte, G., Monaco, M.F., 2007. A tabu search heuristic for the quay crane scheduling problem, Journal of Scheduling 10, 327–336.

[26] Wang, F., Lim, A., 2007. A stochastic beam search for the berth allocation problem, Decision Support Systems 42, 2186–2196.

[27] Wiegmans, B.W., Stekelenburg, D.T., Versteegt, C., Bontekoning, Y.M., 2006. Modeling rail-rail exchange operations: An analysis of conventional and new-generation terminals, Transportation Journal 2006(3), 5–20.

[28] Winter, T., Zimmermann, U.T., 2000. Real-time dispatch of trams in storage yards, Annals of Operations Research 96, 287–315.

[29] Zhu, Y., Lim, A., 2006. Crane scheduling with non-crossing constraints, Journal of Operational Research Society 57, 1472–1481.